

명세서

발명의 명칭: 동형 암호문에 대한 통계 연산 수행하는 장치 및 방법

기술분야

- [1] 본 개시는 동형 암호문에 대한 통계 연산 수행하는 장치 및 방법에 관한 것으로, 구체적으로, 동형 암호문에 대해서 통계 연산이 효율적으로 수행할 수 있는 전자 장치 및 방법에 관한 것이다.

배경기술

- [2] 전자 및 통신 기술의 발달에 힘입어 다양한 장치들 간에 데이터를 송수신하여 활용하는 다양한 서비스가 지원되고 있다. 그 중 일 예로, 사용자가 자신의 개인 정보 등을 서버에 저장하여 두고, 서버의 정보를 이용하는 클라우드 컴퓨팅 서비스도 활발하게 사용되고 있다.
- [3] 이러한 환경에서는 데이터 유출을 방지하기 위한 보안 기술의 사용이 필수적이다. 따라서, 서버는 암호화된 데이터를 저장하게 된다. 이 경우, 서버는 저장된 데이터를 검색하거나 그 데이터에 기초한 일련의 작업을 할 때마다, 암호화된 데이터를 복호화하여야 하므로, 자원 및 시간 낭비가 발생하게 된다.
- [4] 또한, 서버에서 연산을 위해 일시적으로 복호화한 상태에서 제3자의 해킹이 이루어지는 경우, 개인 정보가 제3자에게 손쉽게 유출될 수 있다는 문제점도 있었다.
- [5] 이러한 문제를 해결하기 위하여 동형 암호화 방법이 연구되고 있다. 동형 암호화 방식을 이용하면, 암호화된 정보를 복호화하지 않고 암호문 자체에서 연산을 하더라도, 평문에 대해 연산한 후 암호화한 값과 동일한 결과를 얻을 수 있다. 따라서, 암호문을 복호화하지 않은 상태에서 각종 연산을 수행할 수 있다.
- [6] 다만, 동형 암호문 상태에서의 연산은 평문 상태에서의 연산보다 많은 연산량이 요구된다는 점에서, 처리 시간이 기존의 평문 연산 방식보다 느리다. 특히, 데이터에 대한 통계적 연산은 평문 상태에서도 많은 연산량이 요구된다는 점에서, 동형 암호문에 대한 통계적 연산을 보다 효율적으로 수행할 수 있는 방법이 요구되었다.

발명의 상세한 설명

기술적 과제

- [7] 따라서 본 개시는 상술한 바와 같은 문제점을 해결하기 위한 고안된 것으로, 동형 암호문에 대해서 통계 연산이 효율적으로 수행할 수 있는 전자 장치 및 방법을 제공하는 데 있다.

과제 해결 수단

- [8] 이상과 같은 목적을 달성하기 위하여, 본 개시의 일 실시 예에 따른 전자 장치는 적어도 하나의 인스트럭션(instruction)을 저장하고, 복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하는 메모리, 및 상기 적어도 하나의

인스트럭션을 실행하는 프로세서를 포함하고, 상기 프로세서는 상기 적어도 하나의 인스트럭션을 실행함으로써, 상기 복수의 동형 암호문에 대한 연산 명령이 수신되면 상기 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성할 수 있다.

- [9] 이 경우, 상기 동형 암호문은 복수의 슬롯을 포함하며, 상기 복수의 슬롯 각각에 하나의 변수 정보를 가질 수 있다.
- [10] 한편, 상기 빈 마스크는 복수의 슬롯을 포함하며, 상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하며, 상기 프로세서는 동형 암호문 각각에 대해서 상기 동형 암호문에 포함된 변수 정보별 복수의 빈 마스크를 생성하고, 생성된 복수의 빈 마스크 중 상기 변수 조합에 대응되는 빈 마스크를 선별하고, 상기 선별된 빈 마스크 간의 곱을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성할 수 있다.
- [11] 한편, 상기 빈 마스크는 복수의 슬롯을 포함하며, 상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하는 복수의 서브 슬롯을 포함하고, 상기 프로세서는 동형 암호문 각각에 대해서 하나의 빈 마스크를 생성하고, 상기 복수의 빈 마스크 중 상기 변수 조합에 대응되는 상기 빈 마스크 내의 서브 슬롯을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성할 수 있다.
- [12] 이 경우, 상기 복수의 서브 슬롯은, 기설정된 비트 간격을 갖고 하나의 슬롯 내에 배치될 수 있다.
- [13] 한편, 상기 프로세서는 동일한 특징에 대한 복수의 정보를 포함하는 제1 동형 암호문 및 제2 동형 암호문을 하나의 동형 암호문으로 결합할 수 있다.
- [14] 이 경우, 상기 프로세서는 상기 제1 동형 암호문과 상기 제2 동형 암호문 내의 공통된 정보에 대한 제1 동형 암호문 내의 제1 위치 정보와 상기 제2 동형 암호문 내의 제2 위치 정보를 이용하여, 상기 제1 동형 암호문과 제2 동형 암호문을 하나로 결합할 수 있다.
- [15] 이 경우, 상기 프로세서는 제1 및 제2 동형 암호문 내에 포함된 복수의 정보 각각에 대해서 기설정된 공통된 키로 한 방향 암호화 방식으로 암호화된 데이터와 상기 암호화된 데이터에 대한 동형 암호문 내의 위치 정보가 입력되면, 상기 제1 동형 암호문에 대한 암호화된 데이터와 상기 제2 동형 암호문에 대한 암호화된 데이터를 비교하여, 두 동형 암호문 간에 공통된 정보를 갖는 상기 제1 위치 정보와 상기 제2 위치 정보를 확인할 수 있다.
- [16] 한편, 본 개시의 일 실시 예에 따른 동형 암호문에 대한 암호문 처리 방법은 복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하고, 상기 복수의 동형 암호문에 대한 연산 명령을 수신하는 단계, 상기 복수의 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 생성하는 단계, 상기 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성하는 단계, 및 상기 생성된 개수 정보를 출력하는 단계를 포함한다.
- [17] 이 경우, 상기 동형 암호문은 복수의 슬롯을 포함하며 상기 복수의 슬롯 각각에

하나의 변수 정보를 가질 수 있다.

- [18] 한편, 상기 빈 마스크는 복수의 슬롯을 포함하며, 상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하며, 상기 빈 마스크를 생성하는 단계는, 동형 암호문 각각에 대해서 상기 동형 암호문에 포함된 변수 정보별 복수의 빈 마스크를 생성하고, 상기 개수 정보를 생성하는 단계는 생성된 복수의 빈 마스크 중 상기 변수 조합에 대응되는 빈 마스크를 선별하고, 상기 선별된 빈 마스크 간의 곱을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성할 수 있다.
- [19] 한편, 상기 빈 마스크는, 복수의 슬롯을 포함하며 상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하는 복수의 서브 슬롯을 포함하고, 상기 빈 마스크를 생성하는 단계는 동형 암호문 각각에 대해서 하나의 빈 마스크를 생성하고, 상기 개수 정보를 생성하는 단계는 상기 복수의 빈 마스크 중 상기 변수 조합에 대응되는 상기 빈 마스크 내의 서브 슬롯을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성할 수 있다.
- [20] 이 경우, 상기 복수의 서브 슬롯은, 기설정된 비트 간격을 갖고 하나의 슬롯 내에 배치될 수 있다.
- [21] 한편, 본 암호화 처리 방법은 동일한 특징에 대한 복수의 정보를 포함하는 제1 동형 암호문 및 제2 동형 암호문을 하나의 동형 암호문으로 결합하는 단계를 더 포함할 수 있다.
- [22] 이 경우, 상기 결합하는 단계는 상기 제1 동형 암호문과 상기 제2 동형 암호문 내의 공통된 정보에 대한 제1 동형 암호문 내의 제1 위치 정보와 상기 제2 동형 암호문 내의 제2 위치 정보를 이용하여, 상기 제1 동형 암호문과 제2 동형 암호문을 하나로 결합할 수 있다.
- [23] 이 경우, 상기 결합하는 단계는 제1 및 제2 동형 암호문 내에 포함된 복수의 정보 각각에 대해서 기설정된 공통된 키로 일방향 암호화 방식으로 암호화된 데이터와 상기 암호화된 데이터에 대한 동형 암호문 내의 위치 정보가 입력되면, 상기 제1 동형 암호문에 대한 암호화된 데이터와 상기 제2 동형 암호문에 대한 암호화된 데이터를 비교하여, 두 동형 암호문 간에 공통된 정보를 갖는 상기 제1 위치 정보와 상기 제2 위치 정보를 확인할 수 있다.
- [24] 한편, 본 개시의 일 실시 예에 따른 암호문 처리 방법을 실행하기 위한 프로그램을 포함하는 컴퓨터 판독가능 기록매체에 있어서, 상기 암호문 처리 방법은 복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하고, 상기 복수의 동형 암호문에 대한 연산 명령을 수신하는 단계, 상기 복수의 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 생성하는 단계, 상기 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성하는 단계, 및 상기 생성된 개수 정보를 출력하는 단계를 포함한다.

발명의 효과

- [25] 이상과 같은 본 개시의 다양한 실시 예들에 따르면, 동형 암호문을 이용하여 다양한 통계 처리가 가능하며, 서로 다른 방식의 데이터 구조를 갖는 동형 암호문에 대해서도 병합하여 통계 처리가 가능하다.

도면의 간단한 설명

- [26] 도 1은 본 개시의 일 실시 예에 따른 네트워크 시스템의 구조를 설명하기 위한 도면,
[27] 도 2는 본 개시의 일 실시 예에 따른 전자 장치의 간단한 구성을 도시한 블록도,
[28] 도 3 및 도 4는 본 개시의 일 실시예에 따른 빈 카운트 연산 방법을 설명하기 위한 도면,
[29] 도 5 및 도 6은 본 개시의 일 실시 예에 따른 확장된 빈 카운트 연산 동작을 설명하기 위한 도면,
[30] 도 7은 복수의 암호화 테이블에 대한 결합 방법을 설명하기 위한 도면,
[31] 도 8은 본 개시의 일 실시 예에 따른 빈 마스크를 이용한 통계 계산 방법을 설명하기 위한 도면,
[32] 도 9 및 도 10은 본 개시의 일 실시 예에 따른 평문을 이용하여 빈 마스크를 생성하는 과정을 설명하기 위한 도면,
[33] 도 11은 본 개시의 일 실시 예에 따른 근사 알고리즘을 도시한 도면,
[34] 도 12는 본 개시의 일 실시 예에 따른 동형 암호문을 이용한 빈 마스크 생성 동작을 설명하기 위한 도면,
[35] 도 13은 본 개시의 일 실시 예에 따른 빈 카운트 연산의 동작을 설명하기 위한 도면,
[36] 도 14는 원본 데이터와 본 개시의 일 실시 예에 따른 라지 빈 카운트 동작의 목표를 설명하기 위한 도면,
[37] 도 15는 빈 마스크를 이용하여 특정 경우의 수의 개수를 산출하는 방법을 설명하기 위한 도면,
[38] 도 16은 본 개시의 일 실시 예에 따른 파워 빈 마스크를 이용한 빈 카운트 연산 동작을 설명하기 위한 도면,
[39] 도 17은 본 개시의 일 실시 예에 따른 에러 항을 고려한 빈 카운트 동작을 설명하기 위한 도면,
[40] 도 18은 본 개시의 일 실시 예에 따른 파워 빈 마스크의 생성 동작을 설명하기 위한 도면,
[41] 도 19는 본 개시의 다른 실시 예에 따른 파워 빈 마스크의 생성 동작을 설명하기 위한 도면,
[42] 도 20은 본 개시의 일 실시 예에 따른 복수의 빈 마스크 간의 곱 연산 동작을 설명하기 위한 도면,
[43] 도 21은 복수의 GPU를 이용한 곱 연산 동작을 설명하기 위한 도면,
[44] 도 22는 본 개시의 일 실시 예에 따른 곱 연산 이후의 복호화 동작을 설명하기

위한 도면,

- [45] 도 23은 본 개시의 일 실시 예에 따른 빈 마스크의 데이터 구조를 도시한 도면,
- [46] 도 24는 본 개시의 일 실시 예에 따른 곱 연산의 결과의 데이터 구조를 도시한 도면,
- [47] 도 25는 본 개시에 따른 비교 동작을 설명하기 위한 도면,
- [48] 도 26 내지 도 28은 본 개시의 일 실시 예에 따른 다양한 통계 산출 방법을 설명하기 위한 도면,
- [49] 도 29는 본 개시에 따른 슬롯 내의 최대 값을 계산하는 동작을 설명하기 위한 도면,
- [50] 도 30은 복수의 블록 내의 여러 칼럼 내의 최대 값을 산출하는 동작을 설명하기 위한 도면,
- [51] 도 31은 본 개시의 일 실시 예에 따른 특정 순위의 값을 산출하는 방법을 도시한 도면, 그리고
- [52] 도 32는 본 개시의 일 실시 예에 따른 암호문 처리 방법을 설명하기 위한 흐름도이다.

발명의 실시를 위한 최선의 형태

- [53] -

발명의 실시를 위한 형태

- [54] 이하에서는 첨부 도면을 참조하여 본 개시에 대해서 자세하게 설명한다. 본 개시에서 수행되는 정보(데이터) 전송 과정은 필요에 따라서 암호화/복호화가 적용될 수 있으며, 본 개시 및 특허청구범위에서 정보(데이터) 전송 과정을 설명하는 표현은 별도로 언급되지 않더라도 모두 암호화/복호화하는 경우도 포함하는 것으로 해석되어야 한다. 본 개시에서 "A로부터 B로 전송(전달)" 또는 "A가 B로부터 수신"과 같은 형태의 표현은 중간에 다른 매개체가 포함되어 전송(전달) 또는 수신되는 것도 포함하며, 반드시 A로부터 B까지 직접 전송(전달) 또는 수신되는 것만을 표현하는 것은 아니다.
- [55] 본 개시의 설명에 있어서 각 단계의 순서는 선행 단계가 논리적 및 시간적으로 반드시 후행 단계에 앞서서 수행되어야 하는 경우가 아니라면 각 단계의 순서는 비제한적으로 이해되어야 한다. 즉, 위와 같은 예외적인 경우를 제외하고는 후행 단계로 설명된 과정이 선행단계로 설명된 과정보다 앞서서 수행되더라도 개시의 본질에는 영향이 없으며 권리범위 역시 단계의 순서에 관계없이 정의되어야 한다. 그리고 본 명세서에서 "A 또는 B"라고 기재한 것은 A와 B 중 어느 하나를 선택적으로 가리키는 것뿐만 아니라 A와 B 모두를 포함하는 것도 의미하는 것으로 정의된다. 또한, 본 개시에서 "포함"이라는 용어는 포함하는 것으로 나열된 요소 이외에 추가로 다른 구성요소를 더 포함하는 것도 포괄하는 의미를 가진다.
- [56] 본 개시에서는 본 개시의 설명에 필요한 필수적인 구성요소만을 설명하며, 본

개시의 본질과 관계가 없는 구성요소는 언급하지 아니한다. 그리고 언급되는 구성요소만을 포함하는 배타적인 의미로 해석되어서는 안 되며 다른 구성요소도 포함할 수 있는 비배타적인 의미로 해석되어야 한다.

- [57] 그리고 본 개시에서 "값"이라 함은 스칼라값뿐만 아니라 벡터, 다항식 형태도 포함하는 개념으로 정의된다.
- [58] 후술하는 본 개시의 각 단계의 수학적 연산 및 산출은 해당 연산 또는 산출을 하기 위해 공지되어 있는 코딩 방법 및/또는 본 개시에 적합하게 고안된 코딩에 의해서 컴퓨터 연산으로 구현될 수 있다.
- [59] 이하에서 설명하는 구체적인 수학적식은 가능한 여러 대안 중에서 예시적으로 설명되는 것이며, 본 개시의 권리 범위가 본 개시에 언급된 수학적식에 제한되는 것으로 해석되어서는 아니된다.
- [60] 설명의 편의를 위해서, 본 개시에서는 다음과 같이 표기를 정하기로 한다.
- [61] $a \leftarrow D$: 분포(D)에 따라서 원소(a)를 선택함
- [62] $s_1, s_2 \in R$: S1, S2 각각은 R 집합에 속하는 원소이다.
- [63] $\text{mod}(q)$: q 원소로 모듈(modular) 연산
- [64] $\lfloor - \rfloor$: 내부 값을 반올림함
- [65] 이하에서는 첨부된 도면을 이용하여 본 개시의 다양한 실시 예들에 대하여 구체적으로 설명한다.
- [66] 도 1은 본 개시의 일 실시 예에 따른 네트워크 시스템의 구조를 설명하기 위한 도면이다.
- [67] 도 1을 참조하면, 네트워크 시스템은 복수의 전자 장치(100-1 ~ 100-n), 제1 서버 장치(200), 제2 서버 장치(300)를 포함할 수 있으며, 각 구성들은 네트워크(10)를 통해 서로 연결될 수 있다.
- [68] 네트워크(10)는 다양한 형태의 유무선 통신 네트워크, 방송 통신 네트워크, 광통신 네트워크, 클라우드 네트워크 등으로 구현될 수 있으며, 각 장치들은 별도의 매개체 없이 와이파이, 블루투스, NFC(Near Field Communication) 등과 같은 방식으로 연결될 수도 있다.
- [69] 도 1에서는 전자 장치가 복수개(100-1 ~ 100-n)인 것으로 도시하였으나, 반드시 복수개의 전자 장치가 사용되어야 하는 것은 아니며 하나의 장치가 사용될 수도 있다. 일 예로, 전자 장치(100-1 ~ 100-n)는 스마트폰, 태블릿, 게임 플레이어, PC, 랩톱 PC, 홈서버, 키오스크 등과 같은 다양한 형태의 장치로 구현될 수 있으며, 이밖에 IoT 기능이 적용된 가전 제품 형태로도 구현될 수 있다.
- [70] 사용자는 자신이 사용하는 전자 장치(100-1 ~ 100-n)를 통해서 다양한 정보를 입력할 수 있다. 입력된 정보는 전자 장치(100-1 ~ 100-n) 자체에 저장될 수도 있지만, 저장 용량 및 보안 등을 이유로 외부 장치로 전송되어 저장될 수도 있다. 도 1에서 제1 서버 장치(200)는 이러한 정보들을 저장하는 역할을 수행하고, 제2 서버 장치(300)는 제1 서버 장치(200)에 저장된 정보의 일부 또는 전부를 이용하는 역할을 수행할 수 있다.

- [71] 각 전자 장치(100-1 ~ 100-n)는 입력된 정보를 동형 암호화하여, 동형 암호문을 제1 서버 장치(200)로 전송할 수 있다.
- [72] 각 전자 장치(100-1 ~ 100-n)는 동형 암호화를 수행하는 과정에서 산출되는 암호화 노이즈, 즉, 에러를 암호문에 포함시킬 수 있다. 예를 들어, 각 전자 장치(100-1 ~ 100-n)에서 생성하는 동형 암호문은, 추후에 비밀 키를 이용하여 복호화하였을 때 메시지 및 에러 값을 포함하는 결과 값이 복원되는 형태로 생성될 수 있다.
- [73] 일 예로, 전자 장치(100-1 ~ 100-n)에서 생성하는 동형 암호문은 비밀 키를 이용하여 복호화 하였을 때 다음과 같은 성질을 만족하는 형태로 생성될 수 있다.
- [74] [수학식 1]
- [75] $\text{Dec}(ct, sk) = \langle ct, sk \rangle = M + e \pmod{q}$
- [76] 여기서 $\langle, \rangle$ 는 내적 연산(usual inner product), ct 는 암호문, sk 는 비밀 키, M 은 평문 메시지, e 는 암호화 에러 값, $\text{mod } q$ 는 암호문의 모듈러스(Modulus)를 의미한다. q 는 스케일링 팩터(scaling factor)(Δ)가 메시지에 곱해진 결과 값 M 보다 크게 선택되어야 한다. 에러 값 e 의 절대값이 M 에 비해서 충분히 작다면, 암호문의 복호화 값 $M+e$ 는 유효숫자연산에서 원래의 메시지를 동일한 정밀도로 대체할 수 있는 값이다. 복호화된 데이터 중에서 에러는 최하위 비트(LSB) 측에 배치되고, M 은 차하위 비트 측에 배치될 수 있다.
- [77] 메시지의 크기가 너무 작거나 너무 큰 경우, 스케일링 팩터를 이용하여 그 크기를 조절할 수도 있다. 스케일링 팩터를 사용하게 되면, 정수 형태의 메시지뿐만 아니라 실수 형태의 메시지까지도 암호화할 수 있게 되므로, 활용성이 크게 증대할 수 있다. 또한, 스케일링 팩터를 이용하여 메시지의 크기를 조절함으로써, 연산이 이루어지고 난 이후의 암호문에서 메시지들이 존재하는 영역, 즉, 유효 영역의 크기도 조절될 수 있다.
- [78] 실시 예에 따라, 암호문 모듈러스 q 는 다양한 형태로 설정되어 사용될 수 있다. 일 예로, 암호문의 모듈러스는 스케일링 팩터 Δ 의 지수승 $q = \Delta^L$ 형태로 설정될 수 있다. Δ 가 2라면, $q = 2^{10}$ 과 같은 값으로 설정될 수 있다.
- [79] 또 다른 예로, 암호문 모듈러스는 복수의 서로 다른 스케일링 팩터들을 곱한 값으로 설정될 수도 있다. 각 팩터들은 유사 범위 이내의 값, 즉, 서로 비슷한 크기의 값으로 설정될 수 있다. 예를 들어, $q = q_1 q_2 q_3 \dots q_x$ 로 설정될 수 있으며, $q_1, q_2, q_3, \dots, q_x$ 각각은 스케일링 인수 Δ 와 비슷한 크기이며 서로 소 관계의 값으로 설정될 수 있다.
- [80] 스케일링 팩터를 이와 같은 방식으로 설정하게 되면, CRT(Chinese Remainder Theorem)에 따라 전체 연산을 복수개의 모듈러스 연산으로 분리하여 진행할 수 있게 되므로, 연산 부담을 경감시킬 수 있다.
- [81] 또한, 서로 비슷한 크기의 팩터를 사용함에 따라, 후술하는 단계에서 라운딩 처리를 수행하였을 때, 앞선 예에서의 결과값과 거의 같은 결과를 얻을 수 있다.

- [82] 제1 서버 장치(200)는 수신된 동형 암호문을 복호화하지 않고, 암호문 상태로 저장할 수 있다.
- [83] 제2 서버 장치(300)는 동형 암호문에 대한 특정 처리 결과를 제1 서버 장치(200)로 요청할 수 있다. 제1 서버 장치(200)는 제2 서버 장치(300)의 요청에 따라 특정 연산을 수행한 후, 그 결과를 제2 서버 장치(300)로 전송할 수 있다. 여기서 특정 연산은 복수의 동형 암호문에 대한 덧셈, 동형 곱 등과 같은 일반적인 연산뿐만 아니라, 통계적 연산 예를 들어, 평균, 빈도 분포, 선형 회귀(linear Regression), 공분산(covariance)등 과 같은 연산일 수도 있다.
- [84] 이때, 제2 서버 장치(300)는 복수의 동형 암호문에 대한 결합 동작을 수행할 수 있다.
- [85] 일 예로, 두 개의 전자 장치(100-1, 100-2)가 전송한 암호문 ct_1 , ct_2 가 제1 서버 장치(200)에 저장된 경우, 제2 서버 장치(300)는 두 전자 장치(100-1, 100-2)로부터 제공된 정보들을 합산한 값을 제1 서버 장치(200)로 요청할 수 있다. 제1 서버 장치(200)는 요청에 따라 두 암호문을 합산하는 연산을 수행한 후, 그 결과 값($ct_1 + ct_2$)을 제2 서버 장치(300)로 전송할 수 있다.
- [86] 동형 암호문의 성질상, 제1 서버 장치(200)는 복호화를 하지 않은 상태에서 연산을 수행할 수 있고, 그 결과 값도 암호문 형태가 될 수 있다. 이때, 제1 서버 장치(200)는 연산 결과에 대한 부트 스트래핑을 수행할 수 있다.
- [87] 제1 서버 장치(200)는 연산 결과 암호문을 제2 서버 장치(300)로 전송할 수 있다. 제2 서버 장치(300)는 수신된 연산 결과 암호문을 복호화하여, 각 동형 암호문들에 포함된 데이터들의 연산 결과값을 획득할 수 있다. 그리고 제1 서버 장치(200)는 사용자 요청에 따라 연산을 수차례 수행할 수 있다.
- [88] 한편, 도 1에서는 제1 전자 장치 및 제2 전자 장치에서 암호화를 수행하고, 제2 서버 장치가 복호화를 수행하는 경우를 도시하였으나, 이에 한정되는 것은 아니다.
- [89]
- [90] 도 2는 본 개시의 일 실시 예에 따른 전자 장치의 구성을 나타낸 블록도이다.
- [91] 도 2를 참조하면, 전자 장치(100)는 메모리(110), 프로세서(120), 통신 장치(130), 디스플레이(140), 및 조작 입력 장치(150)를 포함할 수 있다. 이러한 전자 장치는 PC(Personal computer), 노트북, 스마트폰, 태블릿, 서버 등 다양한 장치일 수 있다.
- [92] 메모리(110)에는 전자 장치(100)에 관한 적어도 하나의 인스트럭션(instruction)이 저장될 수 있다. 예를 들어, 메모리(110)에는 본 개시의 다양한 실시 예에 따라 전자 장치(100)가 동작하기 위한 각종 프로그램(또는 소프트웨어)이 저장될 수 있다.
- [93] 이러한 메모리(110)는 RAM 이나 ROM, Buffer, 캐쉬(Cache), 플래시 메모리, HDD, 외장 메모리, 메모리 카드 등과 같은 다양한 형태로 구현될 수 있으며, 어느 하나로 한정되는 것은 아니다.
- [94] 메모리(110)는 암호화할 메시지를 저장할 수 있다. 여기서 메시지는 사용자가

각종 인용한 각종 신용 정보, 개인 정보 등일 수 있으며, 전자 장치(100)에서 사용되는 위치 정보, 인터넷 사용 시간 정보 등 사용 이력 등과 관련된 정보일 수도 있다.

- [95] 그리고 메모리(110)는 공개 키를 저장할 수 있으며, 전자 장치(100)가 직접 공개 키를 생성한 경우, 비밀 키뿐만 아니라, 공개 키 및 비밀 키 생성에 필요한 각종 파라미터를 저장할 수 있다.
- [96] 그리고 메모리(110)는 후술한 과정에서 생성된 동형 암호문을 저장할 수 있다. 그리고 메모리(110)는 외부 장치에서 전송한 동형 암호문을 저장할 수도 있다. 또한, 메모리(110)는 후술하는 연산 과정에서의 결과물인 연산 결과 암호문을 저장할 수도 있다.
- [97] 통신 장치(130)는 전자 장치(100)를 외부 장치(미도시)와 연결하기 위해 형성되고, 근거리 통신망(LAN: Local Area Network) 및 인터넷망을 통해 외부 장치에 접속되는 형태뿐만 아니라, USB(Universal Serial Bus) 포트 또는 무선 통신(예를 들어, WiFi 802.11a/b/g/n, NFC, Bluetooth) 포트를 통하여 접속되는 형태도 가능하다. 이러한 통신 장치(130)는 송수신부(transceiver)로 지칭될 수도 있다.
- [98] 통신 장치(130)는 공개 키를 외부 장치로부터 수신할 수 있으며, 전자 장치(100) 자체적으로 생성한 공개 키를 외부 장치로 전송할 수 있다.
- [99] 그리고 통신 장치(130)는 외부 장치로부터 메시지를 수신할 수 있으며, 생성한 동형 암호문 또는 연산 결과를 외부 장치로 송신할 수 있다.
- [100] 또한, 통신 장치(130)는 암호문 생성에 필요한 각종 파라미터를 외부 장치로부터 수신할 수 있다. 한편, 구현시에 각종 파라미터는 후술하는 조작 입력 장치(150)를 통하여 사용자로 부터 직접 입력받을 수 있다.
- [101] 또한, 통신 장치(130)는 외부 장치로부터 동형 암호문에 대한 연산을 요청받을 수 있으며, 그에 따라 계산된 결과를 외부 장치에 전송할 수 있다. 여기서 요청받은 연산은 덧셈, 뺄셈, 곱셈(예를 들어, 모듈러 곱셈 연산)과 같은 연산일 수 있으며, 통계적 연산일 수도 있다. 여기서 모듈러 곱셈 연산이란 q 원소로 모듈(modular) 연산하는 것을 의미한다.
- [102] 디스플레이(140)는 전자 장치(100)가 지원하는 기능을 선택받기 위한 사용자 인터페이스 창을 표시한다. 예를 들어, 디스플레이(140)는 전자 장치(100)가 제공하는 각종 기능을 선택받기 위한 사용자 인터페이스 창을 표시할 수 있다. 이러한 디스플레이(140)는 LCD(liquid crystal display), OLED(Organic Light Emitting Diodes) 등과 같은 모니터일 수 있으며, 후술할 조작 입력 장치(150)의 기능을 동시에 수행할 수 있는 터치 스크린으로 구현될 수도 있다.
- [103] 디스플레이(140)는 비밀 키 및 공개 키 생성에 필요한 파라미터의 입력을 요청하는 메시지를 표시할 수 있다. 그리고 디스플레이(140)는 암호화 대상이 메시지를 선택하는 메시지를 표시할 수 있다. 한편, 구현시에 암호화 대상은 사용자가 직접 선택할 수도 있고, 자동으로 선택될 수 있다. 즉, 암호화가 필요한

개인 정보 등은 사용자가 직접 메시지를 선택하지 않더라도 자동으로 설정될 수 있다.

- [104] 조작 입력 장치(150)는 사용자로부터 전자 장치(100)의 기능 선택 및 해당 기능에 대한 제어 명령을 입력받을 수 있다. 예를 들어, 조작 입력 장치(150)는 사용자로부터 비밀 키 및 공개 키 생성에 필요한 파라미터를 입력받을 수 있다. 또한, 조작 입력 장치(150)는 사용자로부터 암호화될 메시지를 설정받을 수 있다.
- [105] 프로세서(120)는 전자 장치(100)의 전반적인 동작을 제어한다. 예를 들어, 프로세서(120)는 메모리(110)에 저장된 적어도 하나의 인스트럭션을 실행함으로써 전자 장치(100)의 동작을 전반적으로 제어할 수 있다. 이러한 프로세서(120)는 CPU(central processing unit), ASIC(application-specific integrated circuit)과 같은 단일 장치로 구성될 수 있으며, CPU, GPU(Graphics Processing Unit) 등의 복수의 구성으로 구성될 수도 있다.
- [106] 프로세서(120)는 전송하고자 하는 메시지가 입력되면 메모리(110)에 저장할 수 있다. 그리고 프로세서(120)는 메모리(110)에 저장된 각종 설정 값 및 프로그램을 이용하여, 메시지를 동형 암호화할 수 있다. 이 경우, 공개 키가 사용될 수 있다.
- [107] 프로세서(120)는 암호화를 수행하는데 필요한 공개 키를 자체적으로 생성하여 사용할 수도 있고, 외부 장치로부터 수신하여 사용할 수도 있다. 일 예로, 복호화를 수행하는 제2 서버 장치(300)가 공개 키를 다른 장치들에게 배포할 수 있다.
- [108] 자체적으로 키를 생성하는 경우, 프로세서(120)는 Ring-LWE 기법을 이용하여 공개 키를 생성할 수 있다. 예를 들면, 프로세서(120)는 먼저 각종 파라미터 및 링을 설정하여, 메모리(110)에 저장할 수 있다. 파라미터의 예로는 평문 메시지 비트의 길이, 공개 키 및 비밀 키의 크기 등이 있을 수 있다.
- [109] 링은 다음과 같은 수학적 식 2로 표현될 수 있다.
- [110] [수학적 식 2]
- [111]
$$R = \frac{Z_q[X]}{f(x)}$$
- [112] 여기서 R은 링, Z_q 는 계수, $f(x)$ 는 n 차 다항식이다.
- [113] 링(Ring)이란 기설정된 계수를 가지는 다항식의 집합으로, 원소들 사이에 덧셈과 곱셈이 정의되어 있으며 덧셈과 곱셈에 대해서 닫혀 있는 집합을 의미한다. 이러한 링은 환으로 지칭될 수 있다.
- [114] 일 예로, 링은 계수가 Z_q 인 n 차 다항식의 집합을 의미한다. 예를 들어, n 이 $\Phi(N)$ 일 때, N 차 사이클로토믹 다항식 (N -th cyclotomic polynomial)을 의미할 수 있다. ($f(x)$)란 $f(x)$ 로 생성되는 $Z_q[x]$ 의 이데알(ideal)을 나타낸다. Euler totient 함수 $\Phi(N)$ 이란 N 과 서로 소이고 N 보다 작은 자연수의 개수를 의미한다. $\Phi_N(x)$ 를 N 차 사이클로토믹 다항식으로 정의하면, 링은 다음과 같은 수학적 식 3으로도 표현될 수 있다. 여기서 N 은 2^l 이 이용될 수 있다.
- [115] [수학적 식 3]

- [116] $R = \frac{Z_q[X]}{\Phi_N(x)}$
- [117] 비밀 키(sk)는 다음과 같이 표현될 수 있다.
- [118] 한편, 상술한 수학식 3의 링은 평문 공간에서 복소수를 갖는다. 한편, 동형 암호문에 대한 연산 속도를 향상하기 위하여, 상술한 링의 집합 중 평문 공간이 실수인 집합만을 이용할 수도 있다.
- [119] 이와 같은 링이 설정되면, 프로세서(120)는 링으로부터 비밀 키(sk)를 산출할 수 있다.
- [120] [수학식 4]
- [121] $sk \leftarrow (1, s(x)), s(x) \in R$
- [122] 여기서, s(x)는 작은 계수로 랜덤하게 생성한 다항식을 의미한다.
- [123] 그리고 프로세서(120)는 링으로부터 제1 랜덤 다항식(a(x))을 산출할 수 있다. 제1 랜덤 다항식은 다음과 같이 표현될 수 있다.
- [124] [수학식 5]
- [125] $a(x) \leftarrow R$
- [126] 또한, 프로세서(120)는 에러를 산출할 수 있다. 예를 들어, 프로세서(120)는 이산 가우시안 분포 또는 그와 통계적 거리가 가까운 분포로부터 에러를 추출할 수 있다. 이러한 에러는 다음과 같이 표현될 수 있다.
- [127] [수학식 6]
- [128] $e(x) \leftarrow D_{\sigma}^n$
- [129] 에러까지 산출되면, 프로세서(120)는 제1 랜덤 다항식 및 비밀 키에 에러를 모듈러 연산하여 제2 랜덤 다항식을 산출할 수 있다. 제2 랜덤 다항식은 다음과 같이 표현될 수 있다.
- [130] [수학식 7]
- [131] $b(x) = -a(x)s(x) + e(x) \pmod{q}$
- [132] 최종적으로 공개 키(pk)는 제1 랜덤 다항식 및 제2 랜덤 다항식을 포함하는 형태로 다음과 같이 설정된다.
- [133] [수학식 8]
- [134] $pk = (b(x), a(x))$
- [135] 상술한 키 생성 방법은 일 예에 불과하므로, 반드시 이에 한정되는 것은 아니며, 이 밖에 다른 방법으로 공개 키 및 비밀 키를 생성할 수도 있음은 물론이다.
- [136] 한편, 프로세서(120)는 공개 키가 생성되면, 다른 장치들에 전송되도록 통신 장치(130)를 제어할 수 있다.
- [137] 그리고 프로세서(120)는 메시지에 대한 동형 암호문을 생성할 수 있다. 예를 들어, 프로세서(120)는 메시지에 대해서 앞서 생성된 공개 키를 적용하여 동형 암호문을 생성할 수 있다.
- [138] 복호화할 메시지는 외부 소스로부터 수신할 수도 있고, 전자 장치(100)에 직접 구비 또는 연결된 입력 장치로부터 입력될 수도 있다. 예를 들어, 전자

- 장치(100)가 터치 스크린이나 키 패드를 포함하는 경우, 프로세서(120)는 사용자가 터치 스크린이나 키 패드를 통해 입력하는 데이터를 메모리(110)에 저장한 후, 암호화할 수 있다. 생성된 동형 암호문은 복호화하였을 때 메시지에 스케일링 팩터를 반영한 값에 에러를 더한 결과값으로 복원되는 형태가 될 수 있다. 스케일링 팩터는 사전에 입력되어 설정된 값을 그대로 사용할 수도 있다.
- [139] 또는, 프로세서(120)는 메시지 및 스케일링 팩터를 승산한 상태에서 바로 공개 키를 이용하여 암호화할 수도 있다. 이 경우, 암호화 과정에서 산출되는 에러가 메시지 및 스케일링 팩터를 승산한 결과값에 가산될 수 있다.
- [140] 또한, 프로세서(120)는 암호문의 길이를 스케일링 팩터의 크기에 대응되도록 생성할 수 있다.
- [141] 그리고 프로세서(120)는 동형 암호문이 생성되면 메모리(110)에 저장하거나, 사용자 요청 또는 기설정된 디폴트 명령에 따라 동형 암호문을 다른 장치에 전송하도록 통신 장치(130)를 제어할 수 있다.
- [142] 한편, 본 개시의 일 실시 예에 따르면, 패킹(packaging)이 이루어질 수도 있다. 동형 암호화에서 패킹을 이용하게 되면, 다수의 메시지를 하나의 암호문으로 암호화하는 것이 가능해진다. 이 경우, 전자 장치(100)에서 각 암호문들 간의 연산을 수행하게 되면, 결과적으로 다수의 메시지에 대해 연산이 병렬적으로 처리되므로 연산 부담이 크게 줄어들게 된다.
- [143] 예를 들어, 프로세서(120)는 메시지가 복수의 메시지 벡터로 이루어지는 경우, 복수의 메시지 벡터를 병렬적으로 암호화할 수 있는 형태의 다항식으로 변환한 후, 그 다항식에 스케일링 팩터를 승산하고 공개 키를 이용하여 동형 암호화할 수도 있다. 이에 따라, 프로세서(120)는 복수의 메시지 벡터를 패킹한 암호문을 생성할 수 있다.
- [144] 한편, 전자 장치(100)가 저장하는 데이터가 통계 테이블인 경우, 프로세서(120)는 동형 암호문의 생성과정에서, 암호문 내의 복수의 슬롯 내에 변수 정보가 포함하는 동형 암호문을 생성할 수 있다. 또한, 프로세서(120)는 동형 암호문의 생성 과정에서, 해당 동형 암호문에 대한 빈 마스크를 생성할 수 있다. 구체적인 빈 마스크 생성동작에 대해서는 도 3과 관련하여 후술한다.
- [145] 그리고 프로세서(120)는 동형 암호문에 대한 복호가 필요한 경우, 동형 암호문에 비밀 키를 적용하여 다항식 형태의 복호문을 생성하고, 다항식 형태의 복호문을 디코딩하여 메시지를 생성할 수 있다. 이때 생성한 메시지는 앞서 설명한 수학식 1에서 언급한 바와 같이 에러를 포함할 수 있다.
- [146] 그리고 프로세서(120)는 암호문에 대해 연산을 수행할 수 있다. 예를 들어, 프로세서(120)는 동형 암호문에 대해서 암호화된 상태를 유지한 상태에서 덧셈, 뺄셈, 또는 곱셈 등의 연산뿐만 아니라, 복수의 정보에 대한 평균, 빈도 분포 등과 같은 다양한 통계 연산을 수행할 수 있다. 구체적인 통계 연산 방법에 대해서는 도 3을 참조하여 후술한다.
- [147] 한편, 전자 장치(100)는 연산이 완료되면, 연산 결과 데이터로부터 유효 영역의

데이터를 검출할 수 있다. 예를 들어, 전자 장치(100)는 연산 결과 데이터를 라운딩 처리를 수행하여 유효 영역의 데이터를 검출할 수 있다.

- [148] 여기서, 라운딩 처리란 암호화된 상태에서 메시지의 반올림(round-off)을 진행하는 것을 의미하며, 다르게는 리스케일링(rescaling)이라고 할 수도 있다. 예를 들어, 전자 장치(100)는 암호문 각각의 성분에 스케일링 인수의 역수인 Δ^{-1} 을 곱하고 반올림하여, 노이즈 영역을 제거할 수 있다. 노이즈 영역은 스케일링 팩터의 크기에 대응되도록 결정될 수 있다. 결과적으로 노이즈 영역이 제외된 유효 영역의 메시지를 검출할 수 있다. 암호화 상태에서 진행되므로 추가적인 에러가 발생하지만 크기는 충분히 작으므로 무시할 수 있다.
- [149] 그리고 상술한 라운딩 처리는 상술한 바와 같은 모듈러 곱셈 연산이 이용될 수 있다.
- [150] 또한, 전자 장치(100)는 연산 결과 암호문 내의 근사 메시지 비중이 임계치를 초과하면, 연산 결과 암호문의 평균 공간을 확장할 수 있다. 예를 들어, 상술한 수식식 1에서 q 가 M 보다 작다면 $M+e(\text{mod } q)$ 는 $M+e$ 와 다른 값을 가지므로 복호화가 불가능해진다. 따라서, q 값은 항상 M 보다 크게 유지되어야 한다. 하지만, 연산이 진행됨에 따라 q 값은 점차 감소하게 된다. 평균 공간의 확장이란 암호문 ct 를 더 큰 모듈러스(modulus)를 가지는 암호문으로 변화시키는 것을 의미한다. 평균 공간을 확장하는 동작은 다르게는 재부팅(rebooting)이라 할 수도 있다. 재부팅을 수행함에 따라, 암호문은 다시 연산이 가능한 상태가 될 수 있다.
- [151] 이상과 같이 본 개시에 따른 전자 장치(100)는 동형 암호문에 대한 연산 동작뿐만 아니라, 복잡한 통계 연산에 대해서도 효율적으로 수행할 수 있다. 또한, 전자 장치(100)는 다수의 장치에서 제공하는 동형 암호문을 하나의 DB로 관리하는 것이 가능하다.
- [152] 이하에서는 동형 암호문에 대한 구체적인 통계 연산 동작을 설명한다.
- [153] 우선적으로 동형 암호문에 대한 효율적인 통계 연산을 위하여, 본 개시에서는 동형 암호문을 다음과 같은 데이터 구조를 갖도록 생성한다.
- [154] 평균의 테이블 데이터, 즉 여러 개의 특징으로 이루어진 다수의 기록 데이터를 어떠한 방식으로 SIMD(Single Instruction Multiple Data) 기능을 제공하는 동형 암호에 저장할지에 대한 내용이다. 이를 위하여, 각 특징별로 데이터를 모아서 암호문에 저장할 수 있다. 즉, 하나의 암호문은 하나의 특징에 속하는 데이터만을 저장할 수 있다. 여기서 하나의 암호문이 하나의 정보만을 포함한다는 것이 아니라, 하나의 암호문이 하나의 특징에 대한 복수의 변수 값을 저장할 수 있다는 의미이다.
- [155] 동형 암호문에서는 다수의 슬롯을 포함할 수 있으며, 각 슬롯에 다수의 정보를 각 슬롯에 저장 가능하다. 따라서, 이 점을 이용하여, 다수의 슬롯 각각에 하나의 특징(즉, 테이블 내의 같은 컬럼 정보)에 대한 값들을 저장할 수 있다.
- [156] 구체적으로, 테이블 데이터를 다음과 같은 형식으로 저장, 관리할 수 있다. 예를 들어, 평균 테이블의 크기가 n, m (여기서 n 은 데이터 행의 길이, m 은 데이터 열의

길이 (=feature의 개수))이고, 완전동형암호문이 포함할 수 있는 데이터의 개수를 $M(= N/2)$ 라고 하면, 암호화된 데이터를 포함하는 암호화 테이블은 다음과 같은 내용을 포함할 수 있다.

- [157] 1. 암호문 $c_{0,0}, c_{0,1}, \dots, c_{0, \lfloor n/M-1 \rfloor}, c_{1,0}, \dots, c_{m-1,0}, c_{m-1,1}, \dots, c_{m-1, \lfloor n/M-1 \rfloor}$ (여기서 암호문 c_{ij} 는 $i+1$ 번째 특징을 담고 있는 $(j-1)$ 번째 암호문이고, 암호문의 개수는 각 feature 별로 $\lfloor n/M \rfloor$ 이고,
- [158] 2. 전체 특징의 개수 m , 데이터 행 n
- [159] 3. 그 외의 부가적인 메타데이터(예를 들어, 각 특징의 이름들, 암호문 1개 (1 Block 이라고도 부른다.) 에 들어가는 데이터 개수, 테이블 이름)
- [160] 이와 같은 방식을 이용하는 경우, 암호화된 테이블 데이터에서 소수의 특징별 통계 값을 계산할 때, 더욱 효율적인 계산이 가능하다. 또한, 서로 다른 방식의 암호화 테이블에 대해서 복호화 없이 테이블 결합을 효율적으로 수행할 수 있다. 테이블 결합 동작에 대해서는 도 7과 관련하여 후술한다.
- [161] 이하에서는 각 동형 암호문이 하나의 특징에 대한 정보를 포함하는 경우를 가정하여, 동형 암호문에 대한 통계 연산 동작을 설명한다.
- [162] 우선적으로, 동형 암호문은 암호화된 상태에서 연산을 수행할 수 있으나, 암호화 과정에서의 연산은 많은 시간이 소요되기 때문에, 효율적인 연산 방식이 필요하다.
- [163] 그리고 통계 처리에 있어서는 특정 조건에 부합하는 변수를 찾고, 찾은 변수에 대한 평균, 분산 등의 처리가 수행된다는 점에서 우선적으로 특정 조건에 부합되는 변수를 찾는 것이 우선적으로 필요하다.
- [164] 이러한 점에서, 이하에서는 특정 조건을 만족하는 변수를 찾기 위한 빈 카운트 연산을 도 3 및 도 4를 참조하여 먼저 설명한다.
- [165]
- [166] 도 3 및 도 4는 본 개시의 일 실시예에 따른 빈 카운트 연산 방법을 설명하기 위한 도면이다. 여기서, 빈 카운트 연산은 두 개 이상의 빈(bin) 변수들의 데이터를 입력으로 하여 데이터 내의 각 변수 값들의 조합에 대한 경우의 수를 세는 역할을 하는 연산이다.
- [167] 도 3을 참조하면, 빈 데이터(310), 중간 데이터(320) 및 결과 데이터(330)가 도시된다. 예를 들어, 도 3과 같이 세 개의 변수(A, B, C) 각각에서의 빈 값들이 각각 $\{1, 2, 3, 4\}$ (A), $\{1, 2, 3, 4\}$ (B), $\{1, 2\}$ (C)인 경우, 세 변수의 조합에 따른 중간 데이터를 생성할 수 있다.
- [168] 이와 같이 중간 데이터를 생성한 이후에, 카운트 값이 있는 중간 데이터를 이용하여 세 변수의 가능한 조합 및 해당 조합에 대한 개수를 나타내는 결과 데이터(330)를 생성할 수 있다.
- [169] 이하에서는 도 4를 참조하여, 상술한 중간 데이터를 생성하는 구체적인 동작을 설명한다.

- [170] 도 4를 참조하면, 먼저 변수, A, B, C의 값이 각 1인 경우의 빈 카운트를 계산하는 과정이다.
- [171] 각 변수(410, 420, 430)는 복수의 슬롯을 포함한다. 도시된 예에서는 8개의 슬롯을 갖는 것으로 도시하였지만, 구현시에는 9개보다 많거나 적은 수의 슬롯을 가질 수 있다.
- [172] 각 변수들에 대해서 각 변수 값에 대한 빈 마스크를 생성할 수 있다. 예를 들어, 변수 A(410)의 경우, 4개의 값을 갖기 때문에, 4개의 값 각각에 대응되는 빈 마스크(420)를 만들 수 있다. 그리고 변수 B(430)의 경우, 4개의 값을 갖기 때문에, 4개의 값 각각에 대한 빈 마스크(420)를 만들 수 있으며, 변수 C(45)의 경우 2개의 값을 갖기 때문에 2개의 값 각각에 대한 빈 마스크(460)를 만들 수 있다.
- [173] 이와 같은 빈 마스크는 암호화 시점에 만들어지거나 테이블 록업 테이블 기능을 이용하여 암호화된 후에 계산하여 빈 마스크를 만들 수 있다. 빈 마스크를 생성하는 보다 구체적인 동작은 도 9 내지 도 12에서 설명한다.
- [174] 이후에, 특정 조합에 해당하는 빈 카운트를 확인시에는 해당 조합에 대응되는 빈 마스크를 선택하고, 선택된 빈 마스크 들의 곱을 통하여 결과를 확인할 수 있다. 예를 들어, 1, 1, 1 조합의 확인이 필요한 경우, 변수 A(420)에 대한 빈 마스크 중 1 값의 빈 마스크(471)와 변수 B(430)에 대한 빈 마스크 중 1 값의 빈 마스크(472)와 변수 C(4500)에 대한 빈 마스크 중 1 값의 빈 마스크(473)를 서로 곱하는 연산을 수행할 수 있다.
- [175] 이와 같은 연산에 의해 생성된 출력 마스크(480)는 해당 조합에 해당하는 위치에 1의 값을 갖게 된다. 이를 통하여 해당 조합을 구성하는 각 변수 내의 위치 및 해당 조합의 개수를 확인할 수 있다.
- [176] 이와 같은 동작을 확장하면 빈 평균, 빈 분산도 계산할 수 있다. 빈 평균은 특정 빈 변수의 조합을 갖는 데이터들의 또 다른 변수에 대한 평균을 얻는 연산이다. 그리고 빈 분산은 데이터들에 대한 분산을 얻는 연산이다. 해당 연산을 수행하는 방법은 도 3의 중간 데이터(320)에 대한 빈 마스크 값을 우리가 평균을 얻고자 하는 다른 변수의 값과 곱하여 원하는 빈 값의 조합을 갖고 있는 행에 대한 값들만 뽑아내도 이들을 모두 더할 수 있다. 그리고 빈 마스크 값을 모두 더하고, 이 두 값을 앞의 값에서 뒤의 값으로 나누어 평균을 구할 수 있다. 이와 같은 방식을 활용하여 분산도 계산을 수행할 수 있다. 더욱 자세한 통계 연산 동작에 대해서는 도 26 내지 도 30에서 설명한다.
- [177] 이와 같은 빈 카운트는 분류 과정에서 이용될 수 있으며, Association rule mining 등의 방법으로 이용할 수 있다. 그러나 분류의 정확도를 높이기 위해서는 분류의 경우의 수가 많아야 하며, 데이터 분석자는 더 많은 빈 값의 범위를 갖는 변수를 많이 조합하여 빈 카운트를 연산하고자 할 수 있다.
- [178] 한편, 다양한 연속 값의 데이터는 분석의 편의상 분위 데이터로 표현 가능한데, 예를 들어, 하나에 대해서 50개의 변수 값이 있을 수 있다.

- [179] 만약, 하나의 변수에 대해서 50개의 변수 값이 있는 경우, 가능한 경우의 수는 상당히 크다. 경우의 수가 상당히 많아지는 경우, 각 변수 값별로 빈 마스크를 생성하여 빈 카운트를 수행한다면, 총 연산 수는 $(n-1) \times w \times \lfloor u/M \rfloor$ (여기서, n 변수 개수, w 는 조합의 개수, M 암호문의 슬롯의 개수, u 전체 데이터의 행의 길이)가 된다. 여기서, w, u 가 수백만 단위고, n 이 10 이하, M 이 수만 단위인 경우, 필요한 곱셈의 횟수는 수억 억 번이 되면, 곱셈 수행 시간을 수 밀리 초로 잡아도 필요한 소요 시간은 수십일 이상 소요가 된다.
- [180] 따라서, 이하에서는 변수의 종류가 다양한 경우의 변형된 빈 카운트 방법에 대해서 설명한다.
- [181] 이하에서의 빈 마스크는 빈 값을 인코딩된 형태로 표현된다. 예를 들어, [1,10]의 빈 값의 범위를 갖는 데이터가 있을 때, 빈 값을 10개의 바이트로 표현할 수 있다. 빈 값 $i(i \in [1,10])$ 를 표현하고 싶다면 10개의 바이트 중 i 번째 바이트를 1로 설정하고 나머지는 0으로 설정하는 방식을 사용할 수 있다.
- [182] 이와 같은 방식으로 설정된 빈 마스크 들끼리는 덧셈할 수 있다. 이에 대해서는 도 5를 참조하여 설명한다.
- [183]
- [184] 도 5는 본 개시의 일 실시 예에 따른 확장된 빈 카운트 연산 동작을 설명하기 위한 도면이다.
- [185] 도 5를 참조하면, 하나의 파워 빈 마스크(502, 512)는 복수의 슬롯을 포함하며, 복수의 슬롯 각각은 복수의 서브 슬롯을 포함할 수 있다. 제1 변수(501)의 경우 4개의 서로 다른 변수 값을 갖는바, 제1 파워 빈 마스크(502)는 4개의 서브 슬롯을 포함할 수 있으며, 각 서브 슬롯은 특정 변수 값의 존재 여부에 대한 정보를 포함할 수 있다. 이러한 파워 빈 마스크의 생성 동작에 대해서는 후술한다.
- [186] 이와 같이 파워 빈 마스크를 활용하는 경우, 앞선 과정에서는 변수 종류별로 개별적인 빈 마스크를 생성하였었어 하나, 파워 빈 마스크는 변수 종류 별로 한 개로 족하다.
- [187] 그리고 연산 방식은 빈 마스크 방식과 동일하며, 특정 조합이 필요한 경우, 해당 조합에 대응되는 파워 빈 마스크의 서브 슬롯을 이용하여 출력 데이터(520)를 생성할 수 있다. 그리고 출력 데이터(520)를 디코딩하여, 각 조합의 값을 산출할 수 있다(530).
- [188] 도 3 및 도 4와 같은 방식은 변수 값별로 개별적인 빈 마스크를 생성한 것이라면, 도 5의 방식은 도 3 및 도 4와 같은 슬롯을 세분화한 서브 슬롯을 활용하여, 복수의 빈 마스크를 하나로 합친 형태라고 할 수 있다.
- [189] 이와 같은 파워 빈 마스크를 활용하는 경우의 이점에 대해서는 도 6을 참조하여 설명한다.
- [190] 도 6을 참조하면, 1~4의 빈 값을 갖는 두 변수(601, 604)에 대해 빈 연산을 수행하며, 기존의 빈 마스크를 이용하는 경우 16번의 곱셈이 필요하다. 그에 따라 16개의 암호문에 대한 복호화를 수행하여야 최종 결과를 알 수 있다.

반면에 확장된 빈 마스크(602)를 사용하는 경우, 4번의 곱셈과 4번의 복호화만으로 결과를 얻어낼 수 있다. 왜냐하면, 곱셈 결과에 변수가 1~4인 경우의 모든 위치 정보가 들어가 있기 때문이다.

- [191] 제안 방법의 특성상 확장된 빈 마스크 간의 곱셈은 동작하지 않기 때문에, 한개의 확장된 빈 마스크만 빈 카운트의 계산에 참여할 수 있다. 또한, 1개의 빈 마스크에 들어갈 수 있는 빈의 개수는 최대 50이므로, 제안 방식을 이용하여 개선할 수 있는 곱셈 횟수의 감소는 1/50이다. 또한, 확장된 빈 마스크를 이용하는 곱셈은 다수개의 빈 정보를 하나의 슬롯에 넣어야 하며, 이는 슬롯당 관리해야 할 비트 길이의 증가를 가져오므로, 이는 곱셈 시간에도 영향을 줄 수 있다. 한편, 상술한 빈의 개수 및 감소 횟수는 예시에 불과하고, 적용되는 동형 암호화 방식에 따라 수치 값은 변경될 수 있다.
- [192] 한편, 통계 연산을 위해서는 하나의 소유자의 테이블뿐만 아니라, 서로 다른 소유자의 테이블을 이용할 필요가 있다.
- [193] 따라서, 이하에서는 서로 다른 장치에서 소유한 테이블을 하나로 결합하는 방법에 대해서 설명한다.
- [194] 서로 다른 DB는 동형 암호화된 데이터를 서로 다른 방식으로 저장하고 있을 수 있다. 그러나 서로 다른 DB 각각이 특징별로 구분하여 테이블의 특징별로 구분하여 저장하는 경우, 손쉽게 두 테이블을 결합하는 것이 가능하다.
- [195] 구체적으로, 암호화된 테이블에 대해서 상이한 제1 전자 장치(100-1), 제2 전자 장치(100-2)가 소유하고 있고, 결합을 위한 키 정보를 서로 공유하고 있다고 가정한다. 만약, 프로토콜을 정직하게 수행하는 제3 전자 장치(100-3)가 있는 경우, 제3 전자 장치(100-3)의 도움으로 두 암호화된 테이블을 결합할 수 있다. 이와 같은 방식은 테이블 암호화를 제외한 추가적인 동형 암호 연산 없이 데이터 결합이 가능하다는 점에서, 효율적으로 수행될 수 있다. 또한, 보안 측면에서도 복수의 테이블의 결합을 수행하는 과정에서 공통된 데이터의 개수(Inner-Join) 또는 상대방이 소유한 데이터 중 자신이 소유하지 않는 데이터의 개수(Outer-join)를 제외하고는 어떠한 정보도 노출되지 않는 장점이 있다.
- [196] 이를 위하여, 결합을 위한 키에 해당하는 칼럼 값들은 동형 암호 대신에 별도의 HMAC 함수를 이용하여 복호화 불가능한 일방향 암호화를 수행할 수 있다. 이때, 각 전자 장치(100-1, 100-2)는 동일한 HMAC 키를 공유하고 있기 때문에, 동일한 키 값에 대한 HMAC 수행 결과는 동일하다. 단 HMAC 결과는 공유키 값에도 의존하기 때문에, 동일한 키 값이라도 다른 HMAC 키를 사용할 경우 다른 HMAC 값이 나오게 된다.
- [197] 이러한 HMAC 값과 해당 HMAC 값을 만들기 위해 사용된 원래의 결합키의 테이블 내의 행의 위치 값을 묶어서 한 쌍의 데이터를 만들고, 이러한 데이터의 집합을 HMAC 키값을 이용하여 정렬한 후 제3 전자 장치(100-3)에 전송할 수 있다. 여기서 위치 값은 결합키 값 및 그와 연결된 다른 데이터들의 행 번호를 의미한다.

- [198] 결합 기관은 HMAC 값을 이용하여 매칭되는 키들을 파악하고, 같이 붙여 있는 행값 정보들을 각 전자 장치가 보내면, 해당 제1, 제2 전자 장치는 해당 행들을 암호화한 데이터를 제3 전자 장치(100-3)에 전달하고, 제3 전자 장치는 두 장치에서 보내온 데이터들의 메타 데이터를 조합하여 하나의 결합된 테이블의 메타 데이터를 만듦으로써, 결합을 수행할 수 있다.
- [199] 보다 자세한 결합 동작은 도 7을 참조하여 이하에서 설명한다.
- [200]
- [201] 도 7은 복수의 암호화 테이블에 대한 결합 방법을 설명하기 위한 도면이다.
- [202] 도 7을 참조하면, 두 전자 장치(100-1, 100-3)가 행과 열의 값이 다른 테이블을 각각 소유하고 있고, 두 테이블을 결합시킬 수 있는 1:1 join에 가까운 결합키(예를 들어, 인명 관련 정보를 담은 두 테이블인 경우 주민등록번호 등)가 양 테이블에 존재할 경우, 두 테이블을 결합한 결합 테이블에 대한 1) 형식의 암호화 테이블을 만드는 프로토콜을 의미한다.
- [203] 프로토콜의 참여자는 다음과 같다. 데이터 소유 기관 D_1, D_2 가 존재하고, 데이터 결합자 F 와 최종적으로 암호화된 결합 테이블을 소유할 Z 가 존재한다. 여기서, D_1, D_2 는 도 1의 도시한 전자 장치 또는 서버가 될 수 있다. 또한, 데이터 결합자도 도 1에 도시한 전자 장치 또는 서버가 될 수 있다.
- [204] 데이터 소유 기관 D_1, D_2 는 동일한 동형암호 암호화키 인스턴스 (pk_h)를 갖고 있고, 또한 동일한 MAC 키 (대칭키 sk_{MAC} : 256 bit random 비트)를 공유하고 있다. 데이터 분석 기관 Z 는 pk_h 로 암호화된 데이터에 대한 계산을 수행할 수 있는 계산키 evk_h 를 갖고 있다.
- [205] 동형암호 관련 파라미터 및 알고리즘은 D_1, D_2, Z 가 공유하고 있고, MAC 알고리즘 ($MAC: \{0,1\}^* \times \{0,1\}^{|sk_{MAC}|} \rightarrow \{0,1\}^{MAC_{len}}$) 은 A, B 가 공유하고 있다.
- [206] 데이터 소유 기관 D_1, D_2 이 소유한 데이터는 다음과 같이 기술될 수 있다.
- [207] 데이터 소유 기관 D_1, D_2 은 각각 $m_{D_1} \times n_{D_1} (m_{D_2} \times n_{D_2})$ 크기의 데이터가 있으며, 첫 번째 feature 는 각 데이터의 소유자의 ID (결합키로 사용)를 나타내며 $id^{D_{1_0}}, \dots, id^{D_{1_{m_{D_1}-1}}} (id^{D_{2_0}}, \dots, id^{D_{2_{m_{D_2}-1}}})$ 로 나타내어질 수 있다.
- [208] 나머지 열에 있는 특징들은 $f^{A_1}, \dots, f^{A_{n_{D_1}-1}} (f^{B_1}, \dots, f^{B_{n_{D_2}-1}})$ 로 나타낼 수 있다.
- [209] 또한 $f^X(id^{X_k}) (i \in [0, m_X-1], k \in [0, n_X-1], X \in \{D_1, D_2\})$ 는 기관 X 가 소유한 id_k 사용자에게 대한 f_i 특징의 값을 의미한다. (여기서, $[a, b]$ 는 a 보다 크거나 같고 b 보다 작거나 같은 모든 정수의 집합이다.)
- [210] 따라서 D_1 기관이 소유한 임의의 사용자 $id^{D_{1_i}}$ 에 대한 데이터 튜플은 $(id^{D_{1_i}} f_1^{D_1}(id^{D_{1_i}}), \dots, f_1^{D_{1_{m_{D_1}-1}}}(id^{D_{1_i}}))$ 로 정의가 될 수 있다.

[211]

[212] 데이터 소유 기관 D_1, D_2 이 공통으로 존재하는 키에 대해서는 결합을 수행하는 Inner-Join 과 두 기관에 있는 모든 데이터에 대해서 결합을 수행하는 Outer-Join으로 나뉘어질 수 있다. 자세한 동작은 아래와 같다.

[213]

[214] (초기 환경) 기관 D_1, D_2, F, Z 가 존재하며, 각각이 소유한 정보는 위의 기술과 동일하다.

[215] 1. 각 기관 $D_X (\in \{1, 2\})$ 는 새로운 열을 추가하여 해당 열에 $MAC_{sk_{MAC}}(id_i^{D_X})$ 를 모든 i, X 에 대해 수행한다. 해당 값을 id^{X_i} 로 대표되는 행에 추가할 수 있다. 해당 값을 $mac_{id^{X_i}}$ 라 기술한다.

[216] 2. 모든 데이터를 $mac_{id^{X_i}}$ 값을 기준으로 행 단위로 정렬할 수 있다. 정렬된 MAC 값을 mac^{X_i} 로 기술하며 해당 위치에 있는 ID 값을 id^{X_i} 라고 기술한다. 즉, 정렬 후 데이터는 $(mac^{X_i}, id^{X_i}, f_1^{X_i}(id^{X_i}), \dots, f_{m_X-1}^{X_i}(id^{X_i})) (i \in [0, n_X-1])$ 형식으로 차례로 저장될 수 있다. 즉 mac_i 가 포함된 행은 입력 테이블의 i 번째 행에 기록되게 된다. 이 테이블을 T_X 라 한다. T_X 는 D_X 가 소유하고 있다.

[217] 3. 각 기관 D_X 는 $L_X = (i, mac_i^X) : i \in [0, m_{D_X}-1]$ 를 1이 순서를 맞추어 F에게 전달할 수 있다.

[218] 4. F는 전달받은 L_1, L_2 를 이용하여 $S_{D_1} = \{mac_i^{D_1} | (i, mac^{D_1}) \in L_1\}$, $S_{D_2} = \{mac_i^{D_2} | (i, mac^{D_2}) \in L_2\}$ 를 계산한 후, $S_{D_1 \cap D_2} = mac^{D_1} \downarrow mac^{D_2} \in S_{D_1}$ and $\exists mac^{D_2} \in S_{D_2}$ such that $mac^{D_1} = mac^{D_2}$ 로 정의한다. 또한 $S_{D_1 - D_2} = S_{D_1} - S_{D_1 \cap D_2}$, $S_{D_2 - D_1} = S_{D_2} - S_{D_1 \cap D_2}$ 로 정의한다. 만약 함수 $index^X : S_{D_X} \rightarrow \mathbb{Z} \cup \{\perp\}$ 를 정의하여 $mac^{D_X} \in S_{D_X}$ 에 대해서 $index^X(mac^{D_X}) = i$ if $(i, mac^{D_X}) \in L_X$ otherwise $\perp$ 로 정의한다. 만약 $index^X(S_{D_X})$ 는 S_{D_X} 의 모든 원소에 대해 각각 $index^X$ 함수를 수행하여 그 결과를 순서가 있는 데이터 형태로 반환하는 것으로 정의한다.

[219] 먼저, Outer-Join 경우부터 기술한다.

[220] a) 순서가 있는 데이터 $P_{D_1} = (S_{D_1 - D_2}, S_{D_1 \cap D_2})$, $P_B = (S_{D_1 \cap D_2}, S_{D_1 - D_2})$ 를 만들어 낸다.

[221] b) $R_{D_1} \leftarrow index^{D_1}(P_{D_1})$, $R_{D_2} \leftarrow index^{D_2}(P_{D_2})$ 를 수행한다.

- [222] c) Permutation π 를 이용하기 위해 난수 seed s_{D_1}, s_{D_2} 를 생성한다. 이를 이용하여 $\pi_{s_{D_1}}(R_{D_1}) \rightarrow \text{UID}_1$, $\pi_{s_{D_2}}(R_{D_2}) \rightarrow \text{UID}_2$ 를 수행하여 R_{D_1}, R_{D_2} 의 값을 섞는다. 여기서 $\pi_s(R)$ 은 R 의 값의 순서를 s 가 제공하는 난수성을 이용하여 순서를 섞은 안전한 permutation 을 의미하며, s 를 모르면 원래의 순서가 있는 데이터 R 에 대한 정보를 알아낼 수 없다. 또한 s 를 알고 있으면 $\pi^{-1}_s(\circ)$ 를 수행할 수 있다.
- [223] d) F는 D_1 에게 UID_1 , D_2 에게 UID_2 를 전달한다. 추가하여 F는 D_1 에게 $|S_{D_2-D_1}|$, D_2 에게 $|S_{D_2-D_1}|$ 를 전달한다.
- [224] e) UID_X 를 전달받은 D_X 는 해당 수열에 있는 값의 순서에 맞추어 데이터를 배열한다. 즉, $\text{UID}_X = \{i_0, i_1, \dots, i_{n_{\text{UID}_X}}\}$ 일 경우, $D_X = (f_1^X(id^{X_{i_1}}), \dots, f_{n_{\text{UID}_X}}^X(id^{X_{i_{n_{\text{UID}_X}}}}))_{i=i_0, i_1, \dots, i_{n_{\text{UID}_X}}}$ 순서로 데이터가 배열된다. 여기서 $n_{\text{UID}_X} = |\text{UID}_X|$ 를 의미한다.
- [225] f) 이후 $(\perp, \dots, \perp)$ 으로 이루어진 길이 m_{D_X} 의 $\perp$ 의 수열을 $|S_{D_X - (\{D_1, D_2\} - D_X)}|$ 개 만큼 D_X 에 붙인다.
- [226] g) X 는 D_X 를 1)의 방법을 이용하여 열별로 암호화하여 암호화 테이블 객체 C^{tb_X} 를 만든 후, 메타 정보와 함께 F에게 전달한다.
- [227] h) F는 각각 D_1, D_2 로부터 전달받은 C^{tb_1}, C^{tb_2} 를 $\pi^{-1}_{s_{D_1}}, \pi^{-1}_{s_{D_2}}$ 를 이용하여 순서를 복구시킨다. 여기서 $\pi^{-1}_{s_{D_1}}, \pi^{-1}_{s_{D_2}}$ 역할은 C^{tb_1}, C^{tb_2} 의 각 원소들의 순서를 입력으로 받아, 결과인 숫자에 해당되는 줄 위치에 암호화된 원소 전체를 보내는 역할을 수행한다. 이때 g) 단계에서 붙여진 데이터에 대한 암호문들은 본 과정을 거치지 않는다.
- [228] i) h) 의 결과를 각각 C'_1, C'_2 라고 하면, 두 테이블을 붙여서 하나의 암호화된 테이블 C' 을 만든다. 이때 결합된 암호화 테이블의 메타데이터는 C'_1, C'_2 각각의 메타정보를 결합하여 생성해 낸다.
- [229] j) F는 결합된 암호화 테이블 C_R 을 Z에게 전달한다.
- [230] Inner-Join 은 위의 Outer-Join 과 유사하나, a), b), d) 과정만 아래와 같이 다르다. 또한 f) 과정은 수행되지 않는다.
- [231] a) 순서가 있는 데이터 $P_{D_1 \cap D_2} = (S_{D_1 \cap D_2})$ 를 만들어 낸다.
- [232] b) $R_{D_1} \leftarrow \text{index}^{D_1}(P_{D_1 \cap D_2})$, $R_{D_2} \leftarrow \text{index}^{D_2}(P_{D_1 \cap D_2})$ 를 수행한다.
- [233] d) F는 D_1 에게 UID_1 , D_2 에게 UID_2 를 전달한다.

- [234] 이하에서는 앞서 설명한 빈 마스크를 이용한 실제 통계 동작에 대해서 설명한다.
- [235]
- [236] 도 8은 본 개시의 일 실시 예에 따른 빈 마스크를 이용한 통계 계산 방법을 설명하기 위한 도면이다.
- [237] 도 8을 참조하면, 로우 데이터와 해당 로우 데이터에 대한 수치 데이터가 표시된다. 로우 데이터에 대해서 수치 데이터를 변환하는 동작은 일반적인바 구체적인 설명은 생략한다. 한편, 도시된 예에서는 설명을 용이하게 하기 위하여, 수치 데이터를 평균 상태로 표시하였지만, 실제 데이터는 동형 암호화된 암호문 상태이다.
- [238] 만약, 상술한 바와 같은 데이터에서 60대 이상의 광역시인 사람의 수를 확인하고자 하는 경우에는 나이에 대해서 3 및 지역에 대해서 2를 갖는 빈 마스크를 이용하여 카운트를 할 수 있다.
- [239] 또한, 서울 사람의 신용 등급 평균을 구해야 하는 경우, 지역에 대해서 1 빈 마스크를 이용하여 해당하는 슬롯을 검출하고, 검출된 슬롯에 대응되는 암호화된 신용 등급의 정보를 동형 덧셈을 수행하여 평균을 산출할 수 있다. 보다 자세한 조건부 통계 동작에 대해서는 이하에서 설명한다.
- [240] 이하에서는 상술한 바와 같은 통계적 계산을 앞서 설명한 빈 마스크 및 확장된 빈 마스크를 이용하여 산출하는 보다 자세한 동작을 설명한다.
- [241] 설명을 용이하게 하기 위하여, 범주형 변수를 ‘빈 변수’ (또는 빈 특징)이라 부르고, 연산의 편의를 위해 빈 특징은 1부터 연속된 양의 정수로 나타낸다. 예를 들어, 카테고리가 3개 있는 빈 변수 한 열은 각 행의 값이 1 또는 2 또는 3이 된다.
- [242] 본 개시에 따른 시스템은 어떤 빈 변수가 특정 값을 갖는, 즉 특정 카테고리에 속하는 데이터에 대한 통계 연산을 제공할 수 있다. 위의 평균, 분산, 표준편차 연산은 모든 데이터에 대한 연산이기 때문에 특정 카테고리에 있는 데이터와 통계적 특성이 다를 수 있기 때문이다.
- [243] 또한, 하나의 빈 변수에 대한 조건뿐 아니라 임의의 수의 여러 빈 변수에 대한 조건을 가진 데이터들에 대해서도 통계적 연산이 가능하다.
- [244] 이상과 같은 통계적 연산은 다음과 같은 연산식을 암호화된 상태에서 구현한 것과 같다.

[245] 평균:

$$E(x^{(f_0)} | v^{(f_0)} = 1) = \sum_{i=0}^{R-1} x^{(f_0)}(i) \cdot v^{(f_0)}(i) / \sum_{i=0}^{R-1} v^{(f_0)}(i) \quad (w/v-bit)$$

$$\begin{cases} E(x^{(f_0)}) = \sum_{i=0}^{R-1} x^{(f_0)}(i) / R & (w/o v-bit) \end{cases}$$

[246]

[247] 분산:

$$Var(x^{(f_0)} | v^{(f_0)} = 1) = E((x^{(f_0)} - E(x^{(f_0)} | v^{(f_0)} = 1))^2 | v^{(f_0)} = 1) \quad (w/v-bit)$$

$$\begin{cases} Var(x^{(f_0)}) = E((x^{(f_0)} - E(x^{(f_0)}))^2) & (w/o v-bit) \end{cases}$$

[248] 상관 계수:

$$[249] \quad r(x^{(f_0)}, x^{(f_1)} | v^{(f_0)}=1, v^{(f_1)}=1) = (w/v-bit)$$

$$\left\{ \frac{E(x^{(f_0)} - E(x^{(f_0)} | cond) | cond) E(x^{(f_1)} - E(x^{(f_1)} | cond) | cond)}{\sqrt{Var(x^{(f_0)} | cond) \sqrt{Var(x^{(f_1)} | cond)}}}, cond = (v^{(f_0)}=1, v^{(f_1)}=1) \right.$$

$$\left. r(x^{(f_0)}, x^{(f_1)}) = \frac{E(x^{(f_0)} - E(x^{(f_0)})) E(x^{(f_1)} - E(x^{(f_1)}))}{\sqrt{Var(x^{(f_0)})} \sqrt{Var(x^{(f_1)})}} \quad (w/ov-bit)$$

[250]

[251] bin 특징들의 인덱스 $f_0, f_1, \dots, f_{m-1}$ 가 있고, 정수 $c_0, c_1, \dots, c_{m-1}$ 이 있다고 하자.

아래 연산은 $x^{(f_0)}=c_0, x^{(f_1)}=c_1, \dots, x^{(f_{m-1})}=c_{m-1}$ 의 bin에 대한 조건을 가지고

통계적 연산을 수행할 수 있다.

[252] [수학식 9]

$$[253] \quad cond = \begin{cases} (x^{(f_0)}=c_0, x^{(f_1)}=c_1, \dots, x^{(f_{m-1})}=c_{m-1}, v^{(f_0)}=1, v^{(f_1)}=1, \dots, v^{(f_{m-1})}=1, v^{(f_i-1)}=1) & (w/v-bit) \\ (x^{(f_0)}=c_0, x^{(f_1)}=c_1, \dots, x^{(f_{m-1})}=c_{m-1}) & (w/ov-bit) \end{cases}$$

[254] Bin Count: $\sum(x^{(f_i)} | cond)$ 데이터 테이블에서 조건을 만족하는 경우의 수를 세는 함수

[255] 평균: $\sum(x^{(f_i)} | cond)$ 조건을 만족하는 $x^{(f_i)}$ 열의 값들의 합

[256] bin 평균: $E(x^{(f_i)} | cond) = \sum(x^{(f_i)} | cond) / cnt(x | cond)$

[257] bin 분산: $Var(x^{(f_i)} | cond) = E((x^{(f_i)} - E(x^{(f_i)} | cond))^2 | cond)$

[258] 보다 구체적인 통계 연산 방법에 대해서는 도 25 내지 도 31에서 설명한다.

[259]

[260] 상술한 과정을 위해서는 bin 변수 열과 특정 정수 값 사이의 비교 연산이 필요하다.

[261] 하지만, 동형 연산을 이용한 비교 연산은 비용이 크기 때문에, 미리 bin 마스크에 비교 결과들을 저장해두고 연산에 활용할 수 있다.

[262] 어떤 bin 특징 f_0 가 최대 bin 변수로 s_0 를 갖는다. 즉 데이터의 한 열 $\overline{x^{(f_0)}}$ 의 각

row는 $\{1, 2, \dots, s_0\}$ 중 한 값을 가질 수 있다. 이때 생성되는 bin 마스크는 총 s_0

개의 암호화된 열로, $b_i^{(f_0)}$ ($i=1, 2, \dots, s_0$)로 표현할 수 있다. 여기서, $b_i^{(f_0)}$ 는

암호화된 한 열로, 각 row는 bin 특징 f_0 의 대응되는 row가 1이면 1, 아니면 0으로

표시한다. 그리고 $b_{ij}^{(f_0)}$ 는 $b_i^{(f_0)}$ 의 j 번째 블록을 의미한다.

[263] 이하에서는 통계적 연산에 이용되는 bin 마스크의 생성 동작을 보다 자세히 설명한다.

[264] 먼저, bin 마스크는 암호화 단계에서 생성할 수도 있으며, 암호화 단계

이후에서도 생성될 수 있다. 먼저, 이하에서는 암호화 단계에서 빈 마스크를 생성하는 동작을 도 9를 참조하여 설명한다.

[265]

[266] 도 9 및 도 10은 본 개시의 일 실시 예에 따른 평문을 이용하여 빈 마스크를 생성하는 과정을 설명하기 위한 도면이다. 구체적으로, 데이터 소유자 측에서 암호화를 진행할 때 평문 상태의 데이터를 참조해 빈 마스크를 생성하는 과정을 설명하기 위한 도면이다. 구체적으로, 도 9는 v-비트 테이블을 이용하는 경우이고, 도 10은 v 비트 테이블을 이용하지 않는 경우이다.

[267] 도 9 및 도 10을 참조하면, 빈 마스크를 만들고자 하는 빈 특징들의 인덱스는 벡터 $\vec{f}=(f_0, f_1, \dots, f_{m-1})$ 로, 각 빈 특징의 최대 빈 값은 $\vec{s}=(s_0, s_1, \dots, s_{m-1})$ 로 나타냈다.

[268] 평문 상태의 데이터 테이블은 $\mathbf{x}$, v-bit 테이블은 $\mathbf{v}$ 로 나타내었다. 출력 값 $\mathbf{b}=\{b^{(f_i)}|i=0,1,\dots,m-1\}$ 이다.

[269]

[270] 이하에서는 암호화 동작 이후에 빈 마스크의 생성 동작을 설명한다.

[271] 암호화 동작 이후의 빈 마스크의 생성은 평문 상태의 데이터에 접근할 수 없을 때, 암호화된 상태에서 동형 연산을 이용해 빈 마스크를 생성할 때 사용할 수 있다.

[272] 먼저, 빈 마스크 생성을 위해 암호화된 상태에서 다음과 같은 정수 비교 연산의 구현이 필요하다.

[273] [수학식 10]

$$[274] \quad (a \equiv b) = \begin{cases} 1 & \text{if } a=b \\ 0 & \text{if } a \neq b \end{cases}$$

[275] 이 연산은 아래와 같은 수학식 11과 같은 함수에 대해서 $(a \equiv b) = \delta(a-b)$ 와 같이 표현할 수 있다.

[276] [수학식 11]

$$[277] \quad \delta(x) = \begin{cases} 1 & \text{if } x=0 \\ 0 & \text{if } x \neq 0 \end{cases}$$

[278]

[279] 범주형 변수는 모두 정수 값을 가지므로, 정수 값 입력에 대하여 $\delta(x)$ 함수를 만족해야 한다. 동형 암호화 시스템에서는 덧셈과 곱셈을 제공하므로, 어떤 다항식으로 $\delta(x)$ 함수를 근사하여 정수 범위에서 이를 만족하도록 한다.

[280] 이때, 근사 함수로 아래와 같은 sinc 함수를 사용할 수 있다.

[281] [수학식 12]

$$[282] \quad \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

[283] 이는 모든 정수 값에서 $\delta(x)$ 와 같은 값을 가지고, 다항식 근사가 가능하다. 하지만, 넓은 범위에서 근사할 경우 근사식의 차수가 커져 계산량이 증가할 수 있다. 이러한 경우, 삼각함수의 배각 공식을 이용해 좁은 범위의 근사로도 넓은 범위에서의 근사할 수 있도록 한다.

[284] 먼저 $\sin 2\phi = 2\sin\phi\cos\phi$ 식을 이용해 sinc 함수를 다음과 같은 수학적 식 13과 같이 바꿀 수 있다.

[285] [수학적 식 13]

$$\begin{aligned} \text{sinc}(x) &= \frac{\sin(\pi x)}{\pi x} = \frac{2}{\pi x} \sin\left(\frac{\pi x}{2}\right) \cos\left(\frac{\pi x}{2}\right) \\ &= \text{sinc}\left(\frac{x}{2^d}\right) \cos\left(\frac{x}{2^d}\right) \cos\left(\frac{x}{2^{d-1}}\right) \dots \cos\left(\frac{x}{2}\right) \end{aligned}$$

[287] $\cos\left(\frac{x}{2^d}\right)$ 값을 알 때, $\cos 2\phi = 2\cos^2\phi - 1$ 식을 이용하면

$$\cos\left(\frac{x}{2^{d-1}}\right), \dots, \cos\left(\frac{x}{2}\right) \text{의 값을 모두 알 수 있으므로, } \text{sinc}\left(\frac{x}{2^d}\right) \text{와 } \cos\left(\frac{x}{2^d}\right)$$

값을 알면 $\text{sinc}(x)$ 값을 구할 수 있다. 따라서 $\left[-\frac{\mu}{2^d}, \frac{\mu}{2^d}\right]$ 의 범위에서의

$\text{sinc}(x)$ 와 $\cos(x)$ 를 근사하는 것으로도 $[-\mu, \mu]$ 에서 $\text{sinc}(x)$ 값을 알 수 있다.

[288] $f(x) = \text{sinc}(x)$ 또는 $f(x) = \cos(x)$ 함수 근사에는 아래와 같은 Chebyshev 근사를 사용할 수 있다. 하지만, 구현 시에는 다른 근사 방식이 이용될 수 있다.

[289] [수학적 식 14]

$$f(x) \approx \frac{1}{2} c_0 + \sum_{j=1}^K c_j T_j(x)$$

[291] 여기에서 c_j 는 수학적 식 15와 같다.

[292] [수학적 식 15]

$$c_j = \frac{2}{K+1} \sum_{k=1}^{K+1} f\left[\cos\left\{\frac{\pi(2k-1)}{2(K+1)}\right\}\right] \cos\left\{\frac{\pi j(2k-1)}{2(K+1)}\right\}$$

[294] 여기서, $T_j(x)$ 는 j 차 Chebyshev 다항식이다.

[295] 이와 같은 근사 방식에 따라 생성된 다항식은 K 차 다항식이다.

[296]

[297] 이와 같은 방식을 이용하면, $[-1, 1]$ 범위에서 적은 오차로 근사를 할 수 있고, K 가 짝수이면 $x=0$ 일 때 근사식이 정확히 1의 값을 갖는다.

[298] 여기에 오차가 있다면 배각 공식을 사용해 $\text{sinc}(x)$ 를 계산할 때 오차가 계속 증폭되어 빈 마스크의 값이 정확히 1을 갖지 못하므로, K 를 짝수로 설정할 수

있다.

- [299] 도 11은 본 개시의 일 실시 예에 따른 근사 알고리즘을 도시한 도면이다. 구체적으로, 도 11의 DiscreteEqualZero 알고리즘은 상술한 방식을 이용한 $\text{sinc}(x)$ 함수의 근사 알고리즘을 나타낸다.
- [300] 도 11을 참조하면, 각 슬롯이 정수 값인 암호문 $\mathbf{x}$ 를 입력으로 받아, 그 대응되는 슬롯의 값이 0일 슬롯은 1, 나머지 슬롯은 0인 암호문을 반환할 수 있다. 위의 근사식을 이용해, $\text{sinc}(x) \approx \sum_{i=0}^K a_i x^{i^*} \cos(\pi x) \approx \sum_{i=0}^K b_i x^i$ 의 계수 $a_i, b_i (i=0, 1, \dots, K)$ 를 미리 계산했다고 가정한다.
- [301] 입력의 μ 에 대해 $\mathbf{x}$ 의 각 슬롯은 $[-\mu, \mu]$ 범위의 값이고, K 는 근사 다항식의 차수로, 짝수이다. 과정의 M 은 한 암호문의 슬롯 개수이다.
- [302] 이때 곱셈 연산은 $O(K+3 \lceil \log_2 \mu \rceil)$ 번이다.
- [303] 상술한 방식을 이용하여 생성한 정수 비교 연산을 이용한 빈 마스크 생성 동작은 도 12를 참조하면 설명한다.
- [304] 도 12는 본 개시의 일 실시 예에 따른 동형 암호문을 이용한 빈 마스크 생성 동작을 설명하기 위한 도면이다.
- [305] 도 12를 참조하면, 빈 마스크를 만들고자 하는 빈 특징들의 인덱스는 벡터 $\vec{f}=(f_0, f_1, \dots, f_{m-1})$ 로, 각 빈 특징의 최대 빈 값은 $\vec{s}=(s_0, s_1, \dots, s_{m-1})$ 로 나타냈다.
- [306] 암호화된 데이터 테이블 X , 암호화된 v-bit 테이블 V 를 입력으로 받을 수 있다.
- [307] 출력 값은 $\mathbf{b}=\{b^{(f_i)} | i=0, 1, \dots, m-1\}$ 이다. 만약 암호화된 상태에서 모든 슬롯의 값이 유효하다면, 도 12의 8, 9번 줄에서 한 번의 곱셈 연산을 줄일 수 있다.
- [308] 이하에서는 생성한 빈 마스크를 이용하여 빈 카운트를 산출하는 구체적인 방법을 설명한다.
- [309] 도 13은 본 개시의 일 실시 예에 따른 빈 카운트 연산의 동작을 설명하기 위한 도면이다.
- [310] 여기서, 빈 카운트 연산은 여러 빈 특징들에 대한 조건을 만족하는 행의 개수를 세는 기능이다.
- [311] 입력으로 빈 특징의 벡터 $\vec{f}=(f_0, f_1, \dots, f_{m-1})$ 와 정수 값으로 이루어진 벡터 $\vec{c}=(c_0, c_1, \dots, c_{m-1})$ 을 받을 때, 빈 카운트 연산은 f_0 번째 특징 값이 $c_0, \dots, f_{m-1}$ 번째 특징 값이 c_{m-1} 의 m 가지 조건을 만족하는 유효한 데이터의 개수를 세는 것이다.
- [312] 이때 앞서 생성한 빈 마스크를 사용하는데, 앞선 빈 마스크 과정에서 v-bit를 참조했기 때문에, v-bit에 대한 별도의 고려는 필요하지 않다.
- [313] 조건에 상응하는 빈 마스크 들을 모두 row-wise하게 곱했을 때 모든 조건을

만족한다면 값이 1이고 아닌 경우 0이 된다.

- [314] 따라서 빈 카운트 연산은 이 곱셈 결과의 모든 슬롯의 값을 더하면 된다.
- [315] 도 13에서 빈 카운트 연산의 입력의 $\mathbf{b} = \{b^{(f_i)} | i=0,1,\dots,m-1\}$ 로, 앞선 빈 카운트 생성 과정에서의 출력 값인 빈 마스크들의 집합이다.
- [316] 그리고 n_b 는 한 열의 블록의 개수, m 은 조건의 개수이며, M 은 한 암호문의 슬롯 수 일 때, 위 과정은 $O(n_b * m)$ 번의 곱셈과 $O(\log 2M)$ 의 회전 연산을 한다.
- [317]
- [318] 이하에서는 상술한 빈 카운트 연산 동작의 변형을 설명한다.
- [319] 먼저, 변형된 빈 카운트 방법(이하에서는 라지 빈 카운트라고 지칭함)은 어떤 데이터의 여러 빈 특징이 만드는 모든 경우의 수를 구해 그 결과를 테이블 형태로 나타내는 것이다.
- [320] 구체적인 동작은 도 14를 참조하여 설명한다.
- [321] 도 14는 원본 데이터와 본 개시의 일 실시 예에 따른 라지 빈 카운트 동작의 목표를 설명하기 위한 도면이다.
- [322] 도 14를 참조하면, 최대 빈 변수가 5개의 빈 특징 A, B, C, D, E에 대해 구하고자 하는 결과 테이블을 보여준다.
- [323] 만약 각각 최대 빈 값으로 $s_0, s_1, \dots, s_{m-1}$ 을 갖는 m 개의 빈 특징에 대해 라지 빈 카운트를 수행하면 모든 $\prod_{i=0}^{m-1} s_i$ 가지 경우의 수를 나타내는 결과 테이블을 얻을 수 있다.
- [324] 빈 카운트 방법은 상술한 바와 같이 미리 만들어진 빈 마스크를 이용한다. 빈 마스크 $\mathbf{b}_i^{(f_o)}$ 는 어떤 빈 특징 f_o 의 각 행의 값이 i 와 일치하는지 (i 는 양의 정수) 여부를 1 또는 0으로 나타낸 암호화된 열이다. (i 와 일치한다면 1, 아니라면 0으로 나타낸다.)
- [325] 도 14에 도시된 바와 같이 5개의 특징이 모두 1이라는 값을 갖는 경우에서의 경우의 수를 세는 방법은 다음과 같다. 그 방법에 대해서는 도 15와 참조하여 설명한다.
- [326] 도 15는 빈 마스크를 이용하여 특정 경우의 수의 개수를 산출하는 방법을 설명하기 위한 도면이다.
- [327] 도 15를 참조하면, 빈 마스크 $\mathbf{b}_1^{(A)}, \mathbf{b}_1^{(B)}, \mathbf{b}_1^{(C)}, \mathbf{b}_1^{(D)}, \mathbf{b}_1^{(E)}$ 를 모두 곱하면, 모든 특징이 1인 경우의 s 행에만 1의 값이 남고, 나머지 경우는 모두 0이 된다.
- [328] 따라서 이 곱셈 결과의 모든 행의 값을 더하면 구하고자 하는 경우의 수를 얻을 수 있다.
- [329] 하지만, 이처럼 빈 마스크들을 곱하고 더하는 과정으로는 모든 조합 중 한 경우의 수만 구할 수 있다.

- [330] 따라서 m 개의 bin 특징 $f_0, f_1, \dots, f_{m-1}$ 이 최대 bin 값으로 $s_0, s_1, \dots, s_{m-1}$ 을 가질 때, 특징들이 이루는 모든 조합의 경우의 수를 구하려면 위의 과정을 $\prod_{i=0}^{m-1} s_i$ 번 반복해야 한다.
- [331] 데이터 테이블이 n 개의 행을 가지고, 각 암호문은 M 개의 슬롯을 가질 때, 각 열은 $\lceil n/M \rceil$ 개의 블록을 가진다. 따라서 총 $\lceil n/M \rceil * (m-1) * \prod_{i=0}^{m-1} s_i$ 번의 곱셈이 필요하다.
- [332] 경우의 수를 구하고자 하는 bin 특징의 수나, 각각의 최대 bin 값이 커질수록 필요한 곱셈 연산의 수도 많아진다.
- [333] 따라서 이하에서는 적은 수의 곱셈으로 경우의 수를 산출하기 위한 방법을 설명한다.
- [334] 이와 같은 목적을 수행하기 위하여, bin 마스크 대신에 파워 bin 마스크(power bin Mask)를 이용할 수 있다.
- [335] 파워 bin 마스크의 각 행은 대응되는 bin 특징의 값이 i 일 때 $2^{(i-1) \times \Delta + \delta}$ 의 값을 갖는다. 즉 각 bin 값마다 Δ 만큼씩의 여유를 주는 것이다.
- [336] 도 16은 본 개시의 일 실시 예에 따른 파워 bin 마스크를 이용한 bin 카운트 연산 동작을 설명하기 위한 도면이다.
- [337] 도 16을 참조하면, $\delta=0, \Delta=3$ 으로 설정하고, 특징 A(1610)를 이용해 파워 bin 마스크(1620)를 만든 예시 상황이다. 각 행은 2진법으로 표현하였다.
- [338] 이와 같이 파워 bin 마스크(1620)가 생성되면, 기존 방법처럼 $p^{(A)}, b_1^{(B)}, b_1^{(C)}, b_1^{(D)}, b_1^{(E)}$ (1630)를 곱할 수 있다.
- [339] 곱셈 결과의 각 행의 값은 특징 B, C, D, E가 모두 1이 아닐 때는 0이 될 것이다. 그리고 모든 행의 값을 더한다. 이때 더하는 행의 개수가 2^Δ 보다 작음이 보장되면, 덧셈 결과는 각각 Δ bit 만큼의 영역 안에 저장된다.
- [340] 도 16의 예시는 제시된 6개의 열(1640)만 더한 결과이다. 이때 가장 하위 3 비트 안의 값은 모든 특징이 1인 경우의 수, 가장 상위 3 비트에 저장된 값은 특징 A가 5이고 나머지는 1인 경우의 수를 보여준다.
- [341] 이 결과 값은 B, C, D, E는 모두 1 이고, $A = 1 \sim 5$ 일때, 5가지 경우의 수를 모두 담고 있다. 따라서 이 방법은 기존 bin 카운트 방법과 같은 수의 곱셈 연산을 수행했지만, 5배 많은 경우의 수를 구한 것과 같다.
- [342] 따라서 m 개의 bin 특징 $f_0, f_1, \dots, f_{m-1}$ 이 최대 bin 값으로 $s_0, s_1, \dots, s_{m-1}$ 을 가지고, f_0 으로 이들의 모든 조합의 경우의 수를 구하고자 할 때, f_0 으로 확장된 bin 마스크를 생성하면, 이론적으로 곱셈의 횟수를 $1/s_0$ 배로 줄일 수 있다.
- [343] 한편, 상술한 바와 같은 파워 bin 마스크 방법을 적용하려면 동형 암호 방식의

에러 항(error term)을 고려하여 빈 마스크의 각 행을 0 또는 1 대신 0 또는 2^{Λ} (여기서 Λ 는 양의 정수)로 표현해야 한다.

[344] 빈 마스크를 $m-1$ 개 곱하면 각 행의 값은 최대 $2^{(m-1)\Lambda}$ 까지 커지는데, 이 값이 동형 암호문의 모듈러스 비트를 넘어서는 안 된다.

[345] 이를 위하여, $m-1$ 개의 빈 특징을 k 개의 새로운 빈 특징으로 표현하는 과정을 포함한다($m-1 > k$).

[346] 이 새로운 k 개의 빈 특징에 대해 위의 방법대로 빈 마스크를 만들고 서로 곱하면 각 행의 최대 값을 $2^{k\Lambda}$ 로 줄일 수 있기 때문이다.

[347] 이와 같은 에러 항을 고려한 전체 빈 카운트 동작은 도 17과 같다.

[348]

[349] 도 17은 본 개시의 일 실시 예에 따른 에러 항을 고려한 빈 카운트 동작을 설명하기 위한 도면이다.

[350] 도 17을 참조하면, B, C, D, E 4개의 빈 특징(1740, 1750)을 2개의 새로운 빈 특징으로 만들고 새로운 빅 빈 마스크를 든다. 이와 같이 빅 빈 마스크가 생성되면, 경우의 수를 구하기 위하여, 앞서 생성한 파워 빅 마스크(1730)에 이를 곱할 수 있다.

[351] 이상과 같은 빈 카운트 연산 동작을 정리하면, 먼저 선행적으로 파워 빈 마스크와 빅 빈 마스크를 생성할 수 있다. 이와 같은 빈 마스크 과정은 암호화 과정에서 수행될 수 있다. 또는 암호화 과정 이후에 수행될 수 있다.

[352] 파워 빈 마스크와 빅 빈 마스크가 생성되면, 해당 마스크 간의 곱셈 동작을 수행할 수 있다. 그리고 연산 결과를 확인하기 위한 복호화 과정을 수행할 수 있다.

[353] 구체적으로, 라지 빈 카운트 연산을 위해 한 개의 특징 (f_0)로는 확장된 빈 마스크를 생성할 수 있다.

[354] 그리고 나머지 $m-1$ 개 ($f_1, f_2, \dots, f_{m-1}$)로는 k 개의 빅 빈 마스크를 생성할 수 있다. ($k < m-1$) 이는 $m-1$ 개의 열을 더 적은 수인 k 개의 열로 나타내고, 이 새로운 열의 빈 마스크를 만드는 과정이다. 아래에는 설명의 편의를 위해 $k=2$ 인 경우를 다룬다. 이렇게 생성한 빅 빈 마스크를 각각 빅 빈 마스크 1과 빅 빈 마스크 2라고 한다.

[355] 이때, 생성된 확장된 빈 마스크, 빅 빈 마스크 1, 빅 빈 마스크 2는 각각 $\mathbf{p}$, $\mathbf{q}^{(i)}$, $\mathbf{r}^{(i)}$ ($i=1, 2, \dots, Q$)로 나타낸다. ($\mathbf{p}$, $\mathbf{q}^{(i)}$, $\mathbf{r}^{(i)}$ ($i=1, 2, \dots, Q$)는 각각 암호화된 한 개의 열과 같다. 따라서 각각 한 열 당 블록 개수 ($\lceil n/M \rceil$) 만큼의 암호문으로 구성된다.)

[356] [수학식 16]

[357] $\mathbf{q} = \{ \mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \dots, \mathbf{q}^{(Q)} \}$, $\mathbf{r} = \{ \mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \dots, \mathbf{r}^{(Q)} \}$

- [358] $p_k^{(i_p)}(j)$, $q_k^{(f_{i_p}, f_{m-p})}(j)$, $r_k^{(f_{i_p}, f_{m-p})}(j)$: 각 마스크의 $k+1$ 번째 암호문의 $j+1$ 번째 슬롯에 해당 하는 값 ($i=0,1,\dots,m-1$, $k=0,1,\dots \lceil n/M \rceil -1$, $j=0,1,\dots,M-1$)
- [359]
- [360] 도 18은 본 개시의 일 실시 예에 따른 파워 빈 마스크의 생성 동작을 설명하기 위한 도면이다.
- [361] 도 18을 참조하면, 확장된 빈 마스크는 각 슬롯의 값은 $2^{(i-1) \times \Delta + \delta}$, $i \in [1, 50]$ 으로 표현된다.
- [362] 한편, 특정 동형 암호화 시스템에서는 에러가 하위 bit에 발생하므로, 파워 빈 마스크의 각 슬롯의 값에 하위 δ bit의 오프셋을 주었다.
- [363] 입력의 x, v, f_0 는 각각 평문 상태의 데이터 테이블, 평문 상태의 v-bit table, 파워 빈 마스크를 만들고자 하는 빈 특징의 인덱스이다.
- [364] 그리고 n 은 데이터 행의 개수, M 은 한 암호문의 슬롯 개수이다.
- [365]
- [366] 도 19는 본 개시의 다른 실시 예에 따른 파워 빈 마스크의 생성 동작을 설명하기 위한 도면이다.
- [367] 도 19를 참조하면, 파워 빈 마스크를 만드는데 사용한 1개를 제외한 $m-1$ 개의 빈 변수들을 2개의 새로운 빈 특징으로 나타냈다고 가정하고, 이 새로운 빈 특징들에 대한 마스크를 생성하는 과정이다.
- [368] 일반적인 빈 마스크는 특정값에 해당 여부를 0 또는 1로 나타내었으나, 이 경우에는 0 또는 2^A 로 나타낸다.
- [369] $m-1$ 개 특징의 인덱스가 각각 $f_1, f_2, \dots, f_{m-1}$ 이고, 각 특징의 최대 빈 값을 $s_1, s_2, \dots, s_{m-1}$ 이라고 할 때, 새로운 두 변수는 각각 $Q = \lceil \sqrt{\prod_{i=1}^{m-1} s_i} \rceil$ 를 최대 빈 변수로 갖는다. 여기서, x, v 는 각각 평문 상태의 data table, 평문 상태의 v-bit table이다.
- [370] 이 과정의 결과로, 가정한 새로운 두 칼럼(column)에 대한 빈 마스크들이 생성될 수 있다. 즉, 행의 개수가 n 인 $2 * Q$ 개의 열이 생성되는 것과 같다.
- [371] 이 과정은 라지 빈 카운트의 곱 과정에서 각 슬롯에 저장된 값이 모듈러스 비트 이상이 되지 않도록 하기 위함이다.
- [372] 이와 같이 라지 빈 카운트와 빅 빈 카운트가 준비되면 곱셈 동작을 수행할 수 있다.
- [373] 구체적으로, 모든 $p_k \cdot q_k^{(i)} \cdot r_k^{(i)}$, ($i, j=1, 2, \dots, Q$, $k=0, 1, \dots \lceil n/M \rceil -1$)를 계산할 수 있다. 이 과정의 동형 곱셈 연산에는 GPU를 사용하고 각 GPU는 블록 단위로 곱셈 연산을 수행할 수 있다. 구체적인 동작은 도 20을 참조하여 이하에서 설명한다.

- [374] 도 20은 본 개시의 일 실시 예에 따른 복수의 빈 마스크 간의 곱 연산 동작을 설명하기 위한 도면이다. 그리고 도 21은 복수의 GPU를 이용한 곱 연산 동작을 설명하기 위한 도면이다.
- [375] 도 20을 참조하면, 각 GPU(2010, 2020, 2030, 2040)는 병렬화되어 작동할 수 있다. 따라서 GPU의 개수가 N_g 개라고 할 때, 각각 1개의 블록씩 총 N_g 개의 블록이 동시에 처리될 수 있다. GPU(2010, 2020, 2030, 2040)들은 처음 N_g 개의 블록의 일을 할당받고, 일을 마치면 그 다음 N_g 개 블록의 일을 차례대로 할당받는 식으로 일을 처리할 수 있다. 그리고 한 번에 GPU에 load 하는 암호문의 개수는 GPU의 메모리 용량에 따라 제한할 수 있다.
- [376] 한 블록을 처리하고 나면 결과 암호문이 Q^2 개가 생성된다.
- [377] 모든 블록에 대해 이를 저장하면 $\lceil n/M \rceil * Q^2$ 개가 되는데 이는 블록 개수가 많아질수록 많은 저장 공간을 요구한다. 따라서 블록 개수가 GPU의 개수 (N_g) 보다 많은 경우, 이전에 처리한 블록의 결과 암호문을 불러와 새로운 결과 암호문들과 더한 후 저장할 수 있다. 따라서 저장되는 암호문의 개수는 최대 $N_g * Q^2$ 로 제한할 수 있다.
- [378] 연산 과정에서 GPU가 한 번에 1개의 확장된 빈 마스크, c 개의 빅 빈 마스크 1, c 개의 빅 빈 마스크2를 load 할 수 있을 때, 알고리즘은 도 21과 같다.
- [379] 도 21에서 c 는 어떤 양의 정수로, 사용하는 GPU의 메모리 크기에 따라 결정된다. 앞의 과정에서 생성한 마스크 칼럼(column)들 (p, q, r)과, 연산을 수행하고자 하는 GPU의 개수를 입력으로 받을 수 있다.
- [380] 이와 같은 과정을 통하여 생성된 마스크는 암호화된 상태이므로, 이 정보를 그대로 활용하는 것이 어렵다. 따라서 이하에서는 곱 연산 동작 이후의 복호화 동작을 설명한다.
- [381] 도 22는 본 개시의 일 실시 예에 따른 곱 연산 이후의 복호화 동작을 설명하기 위한 도면이다.
- [382] 도 22를 참조하면, 먼저 앞선 동작의 출력 값인 $mres = \{res_k^{(i,j)} | k=0, 1 \dots N_g-1, i, j=1, 2 \dots Q\}$ 의 원소를 살펴보면, BigBin1의 값 i , BigBin2의 값 j 에 대해 (i, j) 의 경우의 수는 총 Q^2 이다.
- [383] BigBin1, BigBin2는 $m-1$ 개의 특징을 2개의 특징으로 나타낸 것이므로, (i, j) 쌍은 각각 $m-1$ 개 빈 특징들의 조합 중 한 경우로 매핑 된다. BigBin1의 값이 i , BigBin2의 값이 j 인 암호문은 N_g 개가 있다.
- [384] 따라서 N_g 개의 암호문들을 모두 복호화한 후, 각 슬롯의 값을 bit 단위로 잘라 $(2 \cdot \Delta + l \cdot \Delta + \delta) \approx (2 \cdot \Delta + (l-1) \cdot \Delta + \delta)$ 의 Δ bit들의 값만 더하면 확장된 빈 마스크를 만든 빈 특징 f_0 값이 1인 경우의 수를 구할 수 있다.

- [385] 도시된 바와 같이 해당 알고리즘의 입력값으로는 앞의 빈 마스크의 곱 과정의 결과가 필요하다. 이 과정의 결과로는 모든 경우의 수를 담고 있는 $Q^{2 \times s_0}$ 크기의 표를 얻게 된다.
- [386] 도 23은 본 개시의 일 실시 예에 따른 빈 마스크의 데이터 구조를 도시한 도면이다. 구체적으로, 도 23은 파워 빈 마스크(2310), 빅 빈 마스크(2320, 2330)를 생성하였을 때, 각각의 슬롯에 값이 차지하는 영역을 표시한 도면이다.
- [387] 도 23을 참조하면, 특징 f_0 의 최대 빈의 값이 s_0 이므로, 파워 빈 마스크는 $2^{(\Delta \cdot 0 + \delta)}, 2^{(\Delta \cdot 1 + \delta)}, \dots, 2^{(\Delta \cdot (s_0 - 1) + \delta)}$ 중 한 값을 갖는다. 그리고 빅 빈 마스크 $q_{k^{(i)}}, r_{k^{(i)}}$ 는 2^Δ 또는 0의 값을 갖는다. 따라서 Δ 번째 bit 한 bit만 0 또는 1로 표시하는 것과 같다.
- [388] 도 24는 본 개시의 일 실시 예에 따른 곱 연산의 결과의 데이터 구조를 도시한 도면이다. 구체적으로, 도 24는 곱 연산 과정의 결과로 얻은 임의의 슬롯의 모듈러스 비트를 사용을 나타낸 것이다.
- [389] 도 24를 참조하면, 같은 GPU에서 처리한 결과들을 더하면 아래의 그림과 같이 Δ bit 만큼의 영역이 모두 파워 빈 마스크를 생성한 빈 특징의 값이 1인 경우의 수를 표현한다고 할 수 있다. 이 값이 커져서 빈 특징의 값이 $l+1$ 인 경우를 나타내는 영역을 침범해서는 안 되므로, 한 GPU가 처리하는 최대 블록 개수인 $\lceil \lceil n/M \rceil / N_g \rceil$ 는 2^Δ 를 넘지 않아야 한다.
- [390] 도 24에 도시된 내용을 수식으로 표현하면 파워 빈 마스크의 한 슬롯은 $2^{(i-1) \times \Delta + \delta} + e, i \in [1, s_0]$ 이고, 빅 빈 마스크 들은 $a \cdot 2^\Delta + e, (a \in \{0, 1\})$ 이다.
- [391] 이 연산을 통해 얻고 싶은 것은 특정 Bin 값에 해당하는지로, 곱셈 결과의 임의의 슬롯의 하위 $(2\Delta + s_0 \cdot \Delta + \delta)$ 번째 bit부터 $(2\Delta + s_0 \cdot \Delta + \delta)$ bit까지의 값을 통해 알 수 있다.
- [392] 임의의 슬롯의 곱셈 결과는 다음과 같은 수학적 식 17과 같이 표현할 수 있다.
- [393] [수학적 식 17]
- [394] $\{2^{(i-1) \times \Delta + \delta} + e\} \cdot \{a_1 \cdot 2^\Delta + e\} \cdot \{a_2 \cdot 2^\Delta + e\}$
- [395] $\approx a_1 a_2 \cdot 2^{((i-1) \cdot \Delta + \delta + 2\Delta)} + \{2^{2\Delta} + 2 \cdot 2^{((i-1) \cdot \Delta + \delta + 2\Delta)}\} \cdot e \quad (a_1 a_2 \in \{0, 1\})$
- [396] 이때, 곱셈 결과의 에러가 원하는 값의 하위 비트를 넘지 않아야 하고, 값의 상위 비트는 모듈러스 비트를 넘어서는 안 된다. 즉, 곱셈 결과의 에러 항 $(2^{2\Delta} + 2^{((s_0-1) \cdot \Delta + \delta + \Delta) + 1}) \cdot e$ 는 $2^{2\Delta + \delta}$ 를 침범할 수 없고, 최대 비트인 $(2\Delta + (s_0 - 1) \cdot \Delta + \delta)$ 가 동형 암호화 시스템에서 사용하는 모듈러스 비트보다 커질 수 없다.
- [397] 여기에 곱셈 이후 다른 블록에서 생성된 결과와 더한다는 점까지 고려하면,

위의 조건에서 Δ bit 만큼씩의 여유가 있도록 식을 조정해야 한다. 그 결과는 다음과 같다. b_m 은 모듈러스 비트이고, error term의 log upper bound를 b_e 라 하자. ($|e| < \text{pow}(2, b_e)$, $b_e < 0$) 아래의 수학식 18은 에러 항에 대한 수학식이고, 수학식 19는 모듈러스 비트에 대한 수학식이다.

[398] [수학식 18]

$$[399] \quad (2^{2\Delta} + 2^{((s_0-1) \cdot \Delta + \delta + \Lambda) + 1}) \cdot e < 2^{2\Delta + \delta - \Delta}.$$

[400] [수학식 19]

$$[401] \quad (2\Delta + s_0 \cdot \Delta + \delta) < b_m$$

[402] 따라서 $\lceil \lceil n/M \rceil / N_g \rceil < 2^\Delta$ 의 조건과 위의 두 부등식을 만족하도록 Δ, δ, Λ 을 설정할 수 있다.

[403] 한편, 상술한 과정에서 m-1개의 특징을 두 개의 열로 표현하여 빅 빈 마스크를 만들었지만, 임의의 자연수 k개의 열로 표현하는 것 또한 가능하다.

[404] k 개의 열로 표현하는 경우에 $Q = \lceil \sqrt[k]{\prod_{i=1}^{m-1} s_i} \rceil$ 로 설정하고, 빅 빈 마스크

생성 과정에서 Q로 k번 반복적으로 나누며 그 나머지를 새로운 열의 빈 값으로 설정할 수 있다.

[405] 이 경우 곱셈의 결과는

$$\begin{aligned} & \{2^{(i-1) \cdot \Delta + \delta} + e\} \cdot \prod_{j=1}^k a_j \cdot 2^\Lambda + e \\ & \approx a \cdot 2^{((i-1) \cdot \Delta + \delta + k\Lambda)} + \{2^{k\Lambda} + k \cdot 2^{((i-1) \cdot \Delta + \delta + (k-1)\Lambda)}\} \cdot e \\ & \quad (a, a_j \in \{0, 1\}) \end{aligned}$$

[406] 이 되고, 위의 에러 항과 모듈러스 비트에 대한 제한 식을 다음과 같이 다시 쓸 수 있다.

[407] [수학식 20]

$$[408] \quad (2^{k\Lambda} + k2^{((s_0-1) \cdot \Delta + \delta + (k-1)\Lambda)}) \cdot e < 2^{2\Delta + \delta - \Delta}$$

[409] 이와 같은 라지 빈 카운트 과정에서는 $\lceil n/M \rceil (Q^2 + Q)$ 번의 동형 곱셈 연산을 사용한다.

[410]

[411] 도 25는 본 개시에 따른 비교 동작을 설명하기 위한 도면이다.

[412] 통계 계산을 산출하기 위해서는 동형 암호문 내의 값 비교가 필요하다. 즉, 동형 암호문 내의 암호문 값이 특정 변수 값에 해당하는지를 확인하는 것이 필요하다. 도 25는 이를 위한 비교 알고리즘이다.

[413] 도 25를 참조하면, 두 개의 변수를 입력받고, 두 변수가 연산을 통한 비교 결과를 산출할 수 있다. 한편, 동형 암호문 상태에서는 앞서 설명한 바와 같은 sinc 함수에 대한 근사 함수를 이용하여 비교 결과를 산출할 수 있다.

[414]

- [415] 도 26 내지 도 28은 본 개시의 일 실시 예에 따른 다양한 통계 산출 방법을 설명하기 위한 도면이다.
- [416] 구체적으로, 도 26은 평균 산출 방법에 대한 알고리즘(2600)이다.
- [417] 도 26을 참조하면, bin 특징의 인덱스 벡터 $\vec{f}=(f_0, f_1, \dots, f_{m-1})$ 와 조건 bin 값을 나타내는 벡터 $\vec{c}=(c_0, c_1, \dots, c_{m-1})$ 가 있을 때, f_0 번째 특징 값이 $c_0, \dots, f_{m-1}$ 번째 특징 값이 c_{m-1} 의 m 개 조건을 만족하는 행들의 특징 f_i 의 평균 값을 계산할 수 있다. 이는 데이터로 f_i 열을 갖고, v-bit로 조건에 맞는 bin 마스크들의 곱을 갖는 열에 대해 앞에서의 average (Avg) 과정을 취하는 것과 같다. 이를 위해 임시의 특징 f_* 에 대해 데이터 열 $X^{(f_*)}$ 과 v-bit 열 $V^{(f_*)}$ 를 정하는 과정을 거친다.
- [418] 전체 과정은 다음과 같다. X, V 는 각각 암호화된 데이터 테이블과 v-bit 테이블, b 는 BinMask의 집합이다. $\vec{f}, \vec{c}$ 는 bin 변수에 대한 조건을 표현하고, f_i 는 평균을 구하고자 하는 열의 index, f_* 는 평균을 구하는 과정에서 임시로 생성하는 열의 인덱스이다. k 는 평균 연산의 ApproxIn의 iteration 수이다.
- [419] 위 과정은 Average 과정에 $O(m)$ 번의 곱셈이 추가되는 것과 같다. 따라서 $O(n_b+2k+m)$ 의 곱셈과 $O(2\log 2(M))$ 번의 회전 처리가 필요하다. (n_b 는 한 열의 block 개수, k 는 average 연산의 inverse 과정에서의 iteration 수, M 은 한 암호문당 슬롯의 개수이다.)
- [420]
- [421] 그리고 도 27은 분산을 산출하는 방법에 대한 알고리즘(2700)을 설명하기 위한 도면이다.
- [422] 도 27을 참조하면, 위의 bin average 과정과 마찬가지로, 이번에는 평균이 아닌 분산을 구한다. 역시 동일하게 임시의 데이터 열 $X^{(f_*)}$ 과 v-bit 열 $V^{(f_*)}$ 을 생성해 앞의 분산 연산을 취할 수 있다. 과정은 다음과 같다. X, Y 는 각각 데이터 테이블과 v-bit 테이블, b 는 BinMask의 집합이다. $\vec{f}, \vec{c}$ 는 bin 변수에 대한 조건을 표현하고, f_i 는 평균을 구하고자 하는 열의 index, f_* 는 평균을 구하는 과정에서 임시로 생성하는 열의 index이다. k 는 평균 연산의 ApproxIn의 iteration 수이다.
- [423] 위 과정은 앞의 Variance 과정에 약 $O(m)$ 의 곱셈 과정이 추가된 것과 같다. 따라서 $O(3n_b+2k+m)$ 의 곱셈과 $O(3\log 2(M))$ 의 회전 연산이 필요하다. (n_b 는 한 열의 block 개수, k 는 average 연산의 inverse 과정에서의 iteration 수, M 은 한 암호문당 slot의 개수이다.)
- [424]
- [425] 도 28은 상관 계수를 산출하는 방법에 대한 알고리즘(2800)을 설명하기 위한

도면이다.

- [426] 데이터 테이블의 두 feature f_0, f_1 사이의 피어슨 상관계수를 연산한다. 이때 v-bit 테이블을 참조해 두 특징의 값이 모두 유효한 행에 대해서만 연산을 수행한다. 두 변수 X, Y에 대한 상관계수 공식을

$$r_{XY} = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2} \sqrt{\sum (Y - \bar{Y})^2}}$$

이용한다. 본 동형 암호 방식으로 암호화된 데이터 테이블, 암호화된 V비트 테이블의 두 특징인 f_0, f_1 의 상관계수를 구하는 알고리즘은 다음과 같다. 연산 과정의 ApproxIn의 iteration 수가 k_2 이고, ApproxSqrtInv의 iteration 수가 k_1 이다. 마찬가지로 모든 슬롯의 값이 유효하다고 보장될 때, v-bit을 사용하지 않는 연산도 가능하다.

[427]

- [428] 상술한 작업에서의 어려운 부분은 작업과 관련된 수의 역수를 찾아내는 것이다. 역수를 찾아내기 어려운 이유는 역계산이 필요한 값의 범위를 설정하고, 그 결과가 범위 내에서 발산하지 않도록 매개 변수를 설정하는 것이고, 근사 알고리즘은 주로 반복 알고리즘으로 구성된다는 점이다. 따라서 결과의 정확도를 위해서 반복 횟수를 증가해야 하나, 반복 횟수가 증가하면 계산 비용이 증가된다는 점에서, 적절한 횟수의 반복을 수행하여야 한다. 한편, 동형 암호문은 에러가 포함된다는 점에서, 일정 횟수의 연산 이후에는 재부팅 동작을 수행하여야 한다.

[429]

[430]

- [431] 도 29는 본 개시에 따른 슬롯 내의 최대 값을 계산하는 동작을 설명하기 위한 도면이다.

- [432] 도 29를 참고하면, 해당 알고리즘(2900)은 동형 암호문 내의 하나의 슬롯의 값을 저장하고, 저장된 값과 다른 슬롯의 값을 순차적으로 비교하여 최대 값을 산출할 수 있다. 비교 동작은 앞서 도시한 비교 알고리즘이 이용될 수 있다.

[433]

- [434] 도 30은 복수의 블록 내의 여러 칼럼 내의 최대 값을 산출하는 동작을 설명하기 위한 도면이다.

- [435] 도 30을 참조하면, 해당 알고리즘(3000)은 먼저, 도 29와 같은 알고리즘을 이용하여 블록별로 가장 높은 값을 갖는 최대 값을 산출하고, 산출된 최대 값 간의 비교를 통하여 복수의 블록 내에서의 최대 값을 산출할 수 있다.

- [436] 이상에서는 최대 값 산출 동작만을 설명하였지만, 비교 과정에서 작은 값을 산출하는 동작을 통하여 최소 값을 산출하는 것도 가능하다.

[437]

- [438] 도 31은 본 개시의 일 실시 예에 따른 특정 순위의 값을 산출하는 방법을 도시한 도면이다.
- [439] 도 31을 참조하면, 백분위 수 알고리즘(3100)은 오름차순으로 데이터를 정렬하는 방식과 유사하다. 우선적으로 정렬 과정을 수행하고, 그에 따라 요청된 백분위 수에 대응되는 값을 산출할 수 있다.
- [440]
- [441] 도 32는 본 개시의 일 실시 예에 따른 암호문 처리 방법을 설명하기 위한 흐름도이다.
- [442] 도 32를 참조하면, 먼저, 복수의 동형 암호문에 대한 통계 계산 명령을 수신한다(S3210). 이와 같은 통계 계산 명령을 특정 값을 갖는 변수의 개수 산출, 특정 조건을 만족하는 값의 평균, 분산 등을 포함할 수 있다.
- [443] 그리고 복수의 동형 암호문은 복수의 변수 정보를 암호화된 상태를 저장하고 있을 수 있다.
- [444] 그리고 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 생성한다(S3220). 이와 같은 빈 마스크 생성은 앞서 설명한 바와 같이 동형 암호문의 생성 과정에서 생성될 수 있으며, 동형 암호문 상태에서 생성될 수도 있다. 그리고 빈 마스크는 슬롯당 하나의 변수 정보만을 갖는 빈 마스크일 수 있으며, 복수의 변수 값의 존재 여부를 포함하는 확장된 빈 마스크 또는 앞서 설명한 파워 빈 마스크, 빅 빈 마스크 등일 수 있다.
- [445] 그리고 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성한다(S3230). 구체적으로, 생성된 빈 마스크의 곱을 이용하여 특정 조건에 맞는 카운트 값을 산출할 수 있다.
- [446] 그리고 산출된 개수 정보를 출력할 수 있다. 이와 같은 출력은 암호화 상태에서 수행될 수 있으며, 해당 정보를 복호화하는 과정을 수행하고, 복호화된 결과로서도 출력될 수 있다.
- [447] 따라서, 본 실시 예에 따른 암호화 처리 방법은 동형 암호문에 대한 효율적인 통계 연산이 가능하다. 도 32과 같은 암호문 처리 방법은 도 2의 구성을 가지는 전자 장치상에서 실행될 수 있으며, 그 밖의 다른 구성을 가지는 전자 장치상에서도 실행될 수 있다.
- [448] 또한, 상술한 바와 같은 암호문 처리방법은 컴퓨터에서 실행될 수 있는 실행 가능한 알고리즘을 포함하는 프로그램으로 구현될 수 있고, 상술한 프로그램은 비일시적 판독 가능 매체(non-transitory computer readable medium)에 저장되어 제공될 수 있다.
- [449] 비일시적 판독 가능 매체란 레지스터, 캐쉬, 메모리 등과 같이 짧은 순간 동안 데이터를 저장하는 매체가 아니라 반영구적으로 데이터를 저장하며, 기기에 의해 판독(reading)이 가능한 매체를 의미한다. 구체적으로는, 상술한 다양한 방법을 수행하기 위한 프로그램들은 CD, DVD, 하드 디스크, 블루레이 디스크, USB, 메모리카드, ROM 등과 같은 비일시적 판독 가능 매체에 저장되어 제공될

수 있다.

- [450] 또한, 이상에서는 본 개시의 바람직한 실시 예에 대하여 도시하고 설명하였지만, 본 개시는 상술한 특정의 실시 예에 한정되지 아니하며, 청구범위에서 청구하는 본 개시의 요지를 벗어남이 없이 당해 개시가 속하는 기술분야에서 통상의 지식을 가진자에 의해 다양한 변형실시가 가능한 것은 물론이고, 이러한 변형실시들은 본 개시의 기술적 사상이나 전망으로부터 개별적으로 이해되어서는 안 될 것이다.

청구범위

- [청구항 1] 전자 장치에 있어서,
적어도 하나의 인스트럭션(instruction)을 저장하고, 복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하는 메모리; 및
상기 적어도 하나의 인스트럭션을 실행하는 프로세서;를 포함하고,
상기 프로세서는,
상기 적어도 하나의 인스트럭션을 실행함으로써, 상기 복수의 동형 암호문에 대한 연산 명령이 수신되면 상기 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성하는 전자 장치.
- [청구항 2] 제1항에 있어서,
상기 동형 암호문은 복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각에 하나의 변수 정보를 갖는 전자 장치.
- [청구항 3] 제1항에 있어서,
상기 빈 마스크는,
복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하며,
상기 프로세서는,
동형 암호문 각각에 대해서 상기 동형 암호문에 포함된 변수 정보별 복수의 빈 마스크를 생성하고, 생성된 복수의 빈 마스크 중 상기 변수 조합에 대응되는 빈 마스크를 선별하고, 상기 선별된 빈 마스크 간의 곱을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성하는 전자 장치.
- [청구항 4] 제1항에 있어서,
상기 빈 마스크는,
복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하는 복수의 서브 슬롯을 포함하고,
상기 프로세서는,
동형 암호문 각각에 대해서 하나의 빈 마스크를 생성하고, 상기 복수의 빈 마스크 중 상기 변수 조합에 대응되는 상기 빈 마스크 내의 서브 슬롯을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성하는 전자 장치.
- [청구항 5] 제4항에 있어서,
상기 복수의 서브 슬롯은, 기설정된 비트 간격을 갖고 하나의 슬롯 내에 배치되는 전자 장치.
- [청구항 6] 제1항에 있어서,
상기 프로세서는,

동일한 특징에 대한 복수의 정보를 포함하는 제1 동형 암호문 및 제2 동형 암호문을 하나의 동형 암호문으로 결합하는 전자 장치.

[청구항 7]

제6항에 있어서,
상기 프로세서는,
상기 제1 동형 암호문과 상기 제2 동형 암호문 내의 공통된 정보에 대한 제1 동형 암호문 내의 제1 위치 정보와 상기 제2 동형 암호문 내의 제2 위치 정보를 이용하여, 상기 제1 동형 암호문과 제2 동형 암호문을 하나로 결합하는 전자 장치.

[청구항 8]

제7항에 있어서,
상기 프로세서는,
제1 및 제2 동형 암호문 내에 포함된 복수의 정보 각각에 대해서 기설정된 공통된 키로 일방향 암호화 방식으로 암호화된 데이터와 상기 암호화된 데이터에 대한 동형 암호문 내의 위치 정보가 입력되면, 상기 제1 동형 암호문에 대한 암호화된 데이터와 상기 제2 동형 암호문에 대한 암호화된 데이터를 비교하여, 두 동형 암호문 간에 공통된 정보를 갖는 상기 제1 위치 정보와 상기 제2 위치 정보를 확인하는 전자 장치.

[청구항 9]

동형 암호문에 대한 암호문 처리 방법에 있어서,
복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하고, 상기 복수의 동형 암호문에 대한 연산 명령을 수신하는 단계;
상기 복수의 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여 갖는 빈 마스크를 생성하는 단계;
상기 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성하는 단계; 및
상기 생성된 개수 정보를 출력하는 단계;를 포함하는 암호문 처리 방법.

[청구항 10]

제9항에 있어서,
상기 동형 암호문은 복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각에 하나의 변수 정보를 갖는 암호문 처리 방법.

[청구항 11]

제9항에 있어서,
상기 빈 마스크는,
복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하며,
상기 빈 마스크를 생성하는 단계는,
동형 암호문 각각에 대해서 상기 동형 암호문에 포함된 변수 정보별 복수의 빈 마스크를 생성하고,
상기 개수 정보를 생성하는 단계는,
생성된 복수의 빈 마스크 중 상기 변수 조합에 대응되는 빈 마스크를 선별하고, 상기 선별된 빈 마스크 간의 곱을 이용하여 상기 변수 조합을

- 갖는 개수 정보를 생성하는 암호문 처리 방법.
- [청구항 12] 제9항에 있어서,
상기 빈 마스크는,
복수의 슬롯을 포함하며,
상기 복수의 슬롯 각각은 하나의 변수 값의 존재 여부에 대한 정보를 포함하는 복수의 서브 슬롯을 포함하고,
상기 빈 마스크를 생성하는 단계는,
동형 암호문 각각에 대해서 하나의 빈 마스크를 생성하고,
상기 개수 정보를 생성하는 단계는,
상기 복수의 빈 마스크 중 상기 변수 조합에 대응되는 상기 빈 마스크 내의 서브 슬롯을 이용하여 상기 변수 조합을 갖는 개수 정보를 생성하는 암호문 처리 방법.
- [청구항 13] 제12항에 있어서,
상기 복수의 서브 슬롯은, 기설정된 비트 간격을 갖고 하나의 슬롯 내에 배치되는 암호문 처리 방법.
- [청구항 14] 제9항에 있어서,
동일한 특징에 대한 복수의 정보를 포함하는 제1 동형 암호문 및 제2 동형 암호문을 하나의 동형 암호문으로 결합하는 단계;를 더 포함하는 암호문 처리 방법.
- [청구항 15] 제14항에 있어서,
상기 결합하는 단계는,
상기 제1 동형 암호문과 상기 제2 동형 암호문 내의 공통된 정보에 대한 제1 동형 암호문 내의 제1 위치 정보와 상기 제2 동형 암호문 내의 제2 위치 정보를 이용하여, 상기 제1 동형 암호문과 제2 동형 암호문을 하나로 결합하는 암호문 처리 방법.
- [청구항 16] 제15항에 있어서,
상기 결합하는 단계는,
제1 및 제2 동형 암호문 내에 포함된 복수의 정보 각각에 대해서 기설정된 공통된 키로 일방향 암호화 방식으로 암호화된 데이터와 상기 암호화된 데이터에 대한 동형 암호문 내의 위치 정보가 입력되면, 상기 제1 동형 암호문에 대한 암호화된 데이터와 상기 제2 동형 암호문에 대한 암호화된 데이터를 비교하여, 두 동형 암호문 간에 공통된 정보를 갖는 상기 제1 위치 정보와 상기 제2 위치 정보를 확인하는 암호문 처리 방법.
- [청구항 17] 암호문 처리 방법을 실행하기 위한 프로그램을 포함하는 컴퓨터 판독가능 기록매체에 있어서,
상기 암호문 처리 방법은,
복수의 변수 정보를 암호화된 상태를 저장하는 동형 암호문을 복수개 저장하고, 상기 복수의 동형 암호문에 대한 연산 명령을 수신하는 단계;

상기 복수의 동형 암호문 각각에 대해서 서로 다른 변수 정보 구분하여
갖는 빈 마스크를 생성하는 단계;
상기 빈 마스크를 이용하여 변수 조합에 대응되는 개수 정보를 생성하는
단계; 및
상기 생성된 개수 정보를 출력하는 단계;를 포함하는 컴퓨터 판독가능
기록매체.